

AD-A239 065



1



DTIC
FI ECR
JUL 22 1991
S D

ENHANCEMENTS TO PCRS

THESIS

Laurie M. Rouillard, Captain, USAF

AFIT/GOR/ENS/91M-14

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

91 7 19 12a

①

AFIT/GOR/ENS/91M-14

DTIC
ELECTE
JUL 22 1991
S D

ENHANCEMENTS TO PCRS

THESIS

Laurie M. Rouillard, Captain, USAF

AFIT/GOR/ENS/91M-14

Approved for public release; distribution unlimited

91-05711



91 7 19 122

GENERAL INSTRUCTIONS FOR COMPLETING SF 298

The Report Documentation Page (RDP) is used in announcing and cataloging reports. It is important that this information be consistent with the rest of the report, particularly the cover and title page. Instructions for filling in each block of the form follow. It is important to **stay within the lines** to meet **optical scanning requirements**.

Block 1. Agency Use Only (Leave blank).

Block 2. Report Date. Full publication date including day, month, and year, if available (e.g. 1 Jan 88). Must cite at least the year.

Block 3. Type of Report and Dates Covered. State whether report is interim, final, etc. If applicable, enter inclusive report dates (e.g. 10 Jun 87 - 30 Jun 88).

Block 4. Title and Subtitle. A title is taken from the part of the report that provides the most meaningful and complete information. When a report is prepared in more than one volume, repeat the primary title, add volume number, and include subtitle for the specific volume. On classified documents enter the title classification in parentheses.

Block 5. Funding Numbers. To include contract and grant numbers; may include program element number(s), project number(s), task number(s), and work unit number(s). Use the following labels:

C - Contract	PR - Project
G - Grant	TA - Task
PE - Program Element	WU - Work Unit Accession No.

Block 6. Author(s). Name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. If editor or compiler, this should follow the name(s).

Block 7. Performing Organization Name(s) and Address(es). Self-explanatory.

Block 8. Performing Organization Report Number. Enter the unique alphanumeric report number(s) assigned by the organization performing the report.

Block 9. Sponsoring/Monitoring Agency Name(s) and Address(es). Self-explanatory.

Block 10. Sponsoring/Monitoring Agency Report Number. (If known)

Block 11. Supplementary Notes. Enter information not included elsewhere such as: Prepared in cooperation with...; Trans. of...; To be published in... When a report is revised, include a statement whether the new report supersedes or supplements the older report.

Block 12a. Distribution/Availability Statement. Denotes public availability or limitations. Cite any availability to the public. Enter additional limitations or special markings in all capitals (e.g. NOFORN, REL, ITAR).

DOD - See DoDD 5230.24, "Distribution Statements on Technical Documents."

DOE - See authorities.

NASA - See Handbook NHB 2200.2.

NTIS - Leave blank.

Block 12b. Distribution Code.

DOD - Leave blank.

DOE - Enter DOE distribution categories from the Standard Distribution for Unclassified Scientific and Technical Reports.

NASA - Leave blank.

NTIS - Leave blank.

Block 13. Abstract. Include a brief (Maximum 200 words) factual summary of the most significant information contained in the report.

Block 14. Subject Terms. Keywords or phrases identifying major subjects in the report.

Block 15. Number of Pages. Enter the total number of pages.

Block 16. Price Code. Enter appropriate price code (**NTIS only**).

Blocks 17. - 19. Security Classifications. Self-explanatory. Enter U.S. Security Classification in accordance with U.S. Security Regulations (i.e., UNCLASSIFIED). If form contains classified information, stamp classification on the top and bottom of the page.

Block 20. Limitation of Abstract. This block must be completed to assign a limitation to the abstract. Enter either UL (unlimited) or SAR (same as report). An entry in this block is necessary if the abstract is to be limited. If blank, the abstract is assumed to be unlimited.

AFIT/GOR/ENS/91M-14

ENHANCEMENTS TO PCRS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Operations Research

Laurie M. Rouillard, B.S.

Captain, USAF

March 1991

Accession For	
NTIS CR&I	✓
DHS TAB	✓
Unannounced	✓
Justification	
By	
Date to	
Availability Codes	
Dist	Avail & Ref Special
A-1	

Approved for public release; distribution unlimited

THESIS APPROVAL

STUDENT: Laurie M. Rouillard

CLASS: GOR-91M

THESIS TITLE: Enhancements to PCRS

DEFENSE DATE: 22 February 1991

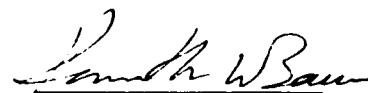
COMMITTEE

NAME/DEPARTMENT

SIGNATURE

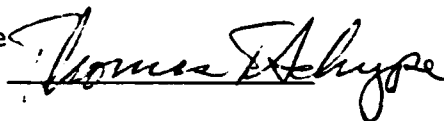
Advisor/co-Advisor
(circle appropriate role)

Major Ken Bauer
AFIT/ENS



Reader

Colonel Tom Schuppe
AFIT/ENS



Preface

The purpose of this study was to extend the capabilities provided by PCRSIM. PCRSIM is a decision support system developed to aid a user in performing response surface methodology. The majority of the software was written in C, which I learned as part of this research effort.

I would like to thank my faculty advisor, Maj Kenneth W. Bauer, for his patience and steady voice in times of need.

Abstract

The purpose of this study was to enhance the capability of PCRSIM, a personal computer based decision support system, to perform response surface methodology. PCRSIM was enhanced to allow the printing of graphical displays such as residual plots and to provide the capability to decode the regression equation back to the original variables. Reverse documentation of the prototype was also developed to describe the interfacing of the individual executable modules and development environment.

Table of Contents

	Page
Preface	ii
List of Figures	v
List of Tables	vi
Abstract	vii
I. Introduction	1
Background	1
Problem Statement	2
Objectives & Related Methods	3
II. Background	5
Experimental Design	5
Design Types	6
Regression	7
Ordinary Least Squares	8
PCRSB	8
PCRSB Experimental Design	9
Design Choices	9
Factor Settings	9
Raw Data Matrix	10
PCRSB Regression Inputs	10
Variable Selection	10
Input Responses	10
Transformations	11
PCRSB Run Regression	11
Aptness Analysis.....	11
Model Results	12
Software Engineering as Applied to PCRSB	12
PCRSB Limitations	13
Summary	14
III. Methodology	15
Documentation	15
Programmatic Flow	15
Maintenance Manual	15
Graphics Capability	16
Variable Decoding	19

IV. Implementation of Methodology	23
Software Environment	23
Software Engineering	
as Applicable to PCRSB	23
Graphics Capability	24
Variable Decoding	25
Assumptions	25
Implementation	27
V. Verification and Validation	30
Graphics Printing	30
Variable Decoding	30
Summary	34
VI. Conclusions and Recommendations	35
Appendix A: PCRSB User's Manual	38
Appendix B: PCRSB Maintenance Manual	48
Appendix C: Test Cases Results	58
Appendix D: Source Listing	67
Appendix E: Optimization Background	291
Bibliography	296
Vita	298

List of Figures

Figure	Page
1. Standardized Residuals vs. \hat{Y}	18
2. Residuals vs. \hat{Y}	18
3. Standardized Residuals vs. Rankits.....	19
4. DECODE FLOWCHART.....	29
5. Executable Module Flowchart.....	48
6. Programmatic Flow	50

List of Tables

Table	Page
1. TEST DATA	31
2. SUMMARIZATION OF TEST CASES	32
3. COMPARISON OF PREDICTED VALUES	33
4. RESULTS FROM TEST CASE 1	60
5. RESULTS FROM TEST CASE 2	61
6. RESULTS FROM TEST CASE 3	63
7. RESULTS FROM TEST CASE 4	64
8. RESULTS FROM TEST CASE 5	66
9. RESULTS FROM TEST CASE 6	67

ENHANCEMENTS TO PCRS

I. Introduction

Background

Response surface methodology is a group of statistical tools used to analyze the relationships between predictors (input variables) and responses (output variables).

Currently it is not easy to conduct response surface methodology because it requires computer assistance and a thorough understanding of the mathematics and statistical theory involved. (Leeper and Meidt, 1990:1)

PCRS, a "micro-computer based decision support system to support response surface methodology analysis" (Leeper and Meidt, 1990:2), was developed by Leeper and Meidt to make response surface methodology analysis easier to perform.

Development of PCRS was envisioned in three main subareas: experimental design, regression and optimization. The experimental design and regression subareas were developed by Leeper and Meidt. PCRS provides the user with several different types of experimental designs. The user selects the appropriate design and runs the regression software to determine the meta-model. A meta-model is a model of a model that simplifies the original model. In this case a meta-model is a polynomial representation of the system being studied. The regression software provides the

user with the tools to assess the aptness of the meta-model. Aptness of a model relates how well the model fits the original system. One such tool is the residual plot, a plot of the model's predicted values on the horizontal axis and the residuals (predicted minus observed) on the vertical axis; this plot is used to determine if the variance is constant, which is one criterion for an apt model.

The creators of PCRSM make several recommendations for improvement in each of the subareas: experimental design and regression. Development of the ability to print the graphical output, and transforming the regression equation back to the original factor setting scale will improve PCRSM. Examination of the current documentation for PCRSM suggests that improved documentation, both within the software and in the form of a maintenance manual, be developed to describe how to rebuild the system.

Problem Statement

It was the purpose of this research to enhance PCRSM with the capability to decode the regression equation, provide the capability to print the graphical displays and develop documentation for PCRSM.

Objectives & Related Methods

The goals of this research were met by accomplishing the following objectives.

Objective 1: Achieve proficiency with PCRSN.

Method: Performed response surface methodology analysis using PCRSN.

Objective 2: Design and implement software routines to allow output of residual plots to a printer.

Methods: The following steps were accomplished to meet this objective.

- Learned C.
- Researched literature on interfacing with printers in C.
- Learned how to integrate output routines with C.
- Developed necessary output routines.
- Tested the output routines by printing the residual plots.
- Validated the printed output plots against the displayed plots.

Objective 3: Design and implement algorithm to transform the regression equation back to the original factor settings scale.

Methods: The following steps were accomplished to meet this objective.

- Developed an algorithm to perform the required transformation.

- Examination of the code for the transformation parameters (High/Low factor settings).
- Retrieved the transformation parameters along with the estimated regression coefficients to be used in the transformation.
- Obtained the decoded regression equation by applying the appropriate transformation.
- Tested the software.

II. Background

"Response surface methodology comprises a group of statistical techniques for empirical model building and model exploitation" (Box and Draper, 1987:3). The main techniques used for response surface methodology are: experimental design, regression, and optimization. These techniques are used to identify the relationship between the inputs and the response, or output (Box and Draper, 1987:1). This chapter will review the techniques of experimental design and regression as well as PCRS, a tool developed to help perform response surface methodology.

Experimental Design

An experimental design provides a systematic method of appropriate settings for all the factors (input variables) of a system (Box and Draper, 1987:17; Myers, 1976:2). The design determines the form of the design matrix used in the regression model. Box and Draper define an experimental design as "a special selected pattern of points chosen to investigate a response function relationship" (Box and Draper, 1987:17).

Experimental designs are discussed in detail by Box and Draper (1987). The order of a design is determined by the degree of the β parameters (see regression) that the collected data can estimate. A first-order design is one

that "permit[s] estimation of the parameters in a planar model" (Mendenhall, 1968:275); a second-order design permits estimation of the β parameters in a quadratic model.

The level of a design describes the number of factor settings on each factor. For example, a two-level design permits two factor level settings: high and low. All two-level designs are first order designs. Three-level designs permit three factor level settings: high, low, and median. Three-level designs are second-order designs. The resolution of a design "is the length of the shortest word in the defining relation" (Box and Draper, 1987:154). Basically the resolution describes the highest degree interaction term that can be estimated.

Design Types. There are several types of experimental designs available some of them are: full factorial, fractional factorial, Central Composite Designs (CCDs), Plackett-Burman, and Box-Behnken designs.

The following first order two-level designs are applicable in PCRS. Full factorial designs requiring 2^k runs, where k is the number of variables in the experiment. These designs exhaust all possible combinations of the factor settings. Fractional factorial designs are designs that require 2^{k-p} runs, where p represents the fraction. A special case of the Plackett-Burman designs implemented in PCRS are defined as follows.

Saturated resolution III two-level designs for up to $k = n-1$ factors in $n = 2^q$ runs can be generated from any 'core' 2^q factorial design by associating new variables with some or all of the interaction columns of the core design. (Box and Draper, 1987:162)

A special type of second order designs are the central composite designs (CCDs). CCDs require five factor settings, the two factor level settings of two-level designs plus as a minimum one center point and two axial points.

Such a design consists of a two level factorial or fractional factorial (chosen so as to allow the estimation of all first-order and two factor interaction terms) augmented with further points which allow pure quadratic effects to be estimated also. (Box and Draper, 1987:305-307)

Box-Behnken designs are second order three-level designs developed by Box and Behnken in 1960 for specific numbers of factors. Box-Behnken designs are used when second order design are desired but the five factor settings required by CCD designs are cumbersome to obtain.

Regression

The technique used in response surface methodology to determine an equation that represents the relationship between the input variables (factors) and the output (the response) is linear regression. The regression equation is used to estimate the response of the system for different factor settings. The regression model in matrix form is

$$Y_{N \times 1} = X_{N \times p} \beta_{p \times 1} + \epsilon_{p \times 1} \quad (1)$$

where

p = number of parameters to be estimated,
 N = number of runs,
 $Y_{N \times 1}$ = vector of responses,
 $\beta_{p \times 1}$ = vector of coefficients (betas),
 $X_{N \times p}$ = design matrix of factor settings,
 $\epsilon_{p \times 1}$ = random error term

The mathematical method used to estimate the beta estimates is ordinary least squares.

Ordinary Least Squares. Ordinary least squares is one technique used to estimate the beta coefficients of the regression model that represents the "best" regression equation. The measure of best is based on the error determined by the "sum of squares of the vertical deviations of the actual points [responses] from the fitted line" (Hillier and Lieberman, 1986:690). Ordinary least squares estimate the regression coefficients such that the sums of squares error is minimized. The least squares estimation of the parameters β -hat given by Neter, Wasserman, and Kutner is:

$$\beta\text{-hat} = (X'X)^{-1}X'Y \quad (2)$$

(Neter, Wasserman, and Kutner, 1985:210)

The ordinary least squares method assumes the following: that the error terms, ϵ_i , are independent, identically distributed with constant variance, σ^2 .

PCRSM

PCRSM is a "micro computer based decision support system to support response surface methodology" (Leeper and Meidt, 1990) developed as a thesis effort at the Air Force

Institute of Technology. PCRSM was envisioned to have three main subareas: experimental design, regression, and optimization. PCRSM currently provides the ability to perform experimental design and regression to determine relationships between the inputs and the response.

PCRSM Experimental Design. The experimental design option of PCRSM provides a user-friendly interactive environment to perform experimental design. This option has three components: design choices, factor settings, and raw data matrix.

Design Choices. The design choices option determines and creates the experimental design based on the number of factors of interest, number of desired center points, number of desired design replications, and whether group or factor screening is to be performed. (Meidt and Bauer, 1990:8)

PCRSM provides six types of two and three level designs. These designs include "full and fractional factorial designs, Plackett-Burman designs, central composite designs, and Box-Behnken designs" (Leeper and Meidt, 1990:77).

Factor Settings. The factor settings option in PCRSM allows the user to provide the high and low values of the factors. "If it's a three level design, the medium value will be calculated as the arithmetic mean of the high and low" (Leeper and Meidt, 1990:78). These factor settings are encoded to take advantage of the "orthogonal or near-orthogonal properties of the resultant design matrix" (Meidt

and Bauer, 1990:5). The encoded variables for a two level design are -1 and 1; and for a three level design are -1, 0, and 1. PCRSM does not require the factor settings to build the design matrix because the design matrix is built from the user inputs from the design inputs section.

Raw Data Matrix. The raw data matrix option creates the design matrix in the original factor settings as specified by the designs option selected. The raw data matrix is not used by PCRSM at this time but is created to be used as input to another regression package if desired.

PCRSM Regression Inputs. The regression inputs option of PCRSM prepares the design matrix and responses for the regression package (Meidt and Bauer, 1990:10) by selection of the appropriate design matrix. The selection of a design matrix lets "the regression model know which design matrix to regress" (Leeper and Meidt, 1990:80). The other options of regression inputs are variable selection, input responses, and transformations.

Variable Selection. The variable selection option of PCRSM allows the user to eliminate insignificant factors from the experimental design.

Input Responses. The input responses option provides the responses to the regression package. "The responses, or outputs, corresponding to the design matrix

can be entered at the keyboard or read in from an ASCII file" (Leeper and Meidt, 1990:80).

Transformations. The transformations option provides several common transformations of the responses. Transformation of the responses is often required to satisfy the constant variance assumption of the least squares method. The transformations available in PCRSM are

- 1) power transformation (y^a)
- 2) natural log of Y
- 3) log base 10 of Y
- 4) arcsin of the square root of Y
- 5) natural log of $(1 + Y)/(1 - Y)$
- 6) inverse of Y (i.e., $1/Y$)
- 7) square root of Y
- 8) square of Y (i.e., Y^2)
- 9) natural log of $(B - Y)$ where B is some value (Leeper and Meidt, 1990:123)

PCRSM Run Regression. The run regression option of PCRSM "performs the regression calculations and generates tabular and graphical displays of the regression statistics" (Meidt and Bauer, 1990). The tabular data can be displayed on the screen, routed to an output file, or sent to a printer. The graphical displays can be viewed only on the screen. This option has two components, aptness analysis and model results.

Aptness Analysis. The aptness analysis option provides three graphical displays that present "information on the standard regression model assumptions (i.e., statistical independence, normality, constant variance)" (Meidt and Bauer, 1990:12). The three displays discussed by

Leeper and Meidt are a scatterplot of the standardized residuals, a scatterplot of the residuals, and a plot of the "ordered standardized residuals versus rankits. The rankits are the expected value of an observation assuming a standard normal distribution" (Leeper and Meidt, 1990:82). The standardized residuals versus rankits plot also contains the Wilk-Shapiro statistic, a test for normality.

Model Results. The model results option of PCRSM provides seven tables containing regression statistics information. The seven tables are

- 1) ANOVA Table
- 2) Coefficient Table
- 3) Variance-Covariance Matrix
- 4) Correlation Matrix
- 5) Lack of Fit ANOVA Table
- 6) Design Matrix
- 7) $(X'X)^{-1}$ Matrix

Appendix A contains the User's Manual developed by Leeper and Meidt modified to incorporate the results of this research effort.

Software Engineering as Applied to PCRSM. Software engineering is defined to be

the establishment and use of sound engineering principles in order to obtain economically software that is reliable and works efficiently on real machines. (Pressman, 1987:19)

PCRSM was developed using prototyping, which is one of the three software engineering techniques that Pressman discusses: classic life cycle, prototyping, and fourth generation techniques. "A prototype is defined as an

original version or model on which a completed software system is formed" (King, 1984:184). In the case of PCRS M the prototype became the final system. In developing a new software system several stages must be met. Some of the most important are determining requirements and documenting the system. "... Additionally these techniques and tools must produce documentation that enables all of these groups to use, understand, change and correct the systems quickly and efficiently" (King, 1984:6). It is in this respect that the design of PCRS M could be improved. The current lack of documentation prevents rapid modification of the system.

The development of PCRS M as described by Leeper and Meidt was performed using prototyping and meetings with an expert to determine the requirements of the system (Leeper and Meidt, 1990:34).

PCRS M Limitations. PCRS M provides the ability to perform response surface methodology and assess the adequacy of the model (see model results above). However, PCRS M does not provide the capability to print the graphical output in support of the model aptness.

PCRS M performs the regression on the design matrix using the encoded values of the factors. The regression equation coefficients provided to the user are in terms of the encoded variables. The output would be easier for the user to interpret if PCRS M converted the regression equation

back to the original factor settings. The resultant decoded coefficients are more meaningful in terms of the original variables.

A major limitation of PCRSM from the software engineering point of view is documentation. PCRSM was well documented with respect to the requirements and theory behind the system but lacks proper documentation for the actual software. This lack of documentation reduces the ability to modify and maintain the system by anyone other than the original system developers.

Summary

Response surface methodology is used to identify the relationship between input factors and the system response. PCRSM was developed as a decision support system to perform response surface methodology. PCRSM provides the user with the ability to specify what type of experimental design to run and performs the regression calculations. The system would be more useful if the capability to print the graphical displays existed and it was possible to convert the regression equation to the original factor settings.

From a software engineering perspective the ability to perform maintenance and provide additional enhancements to PCRSM would be facilitated by more exhaustive documentation.

III. Methodology

This chapter describes the methodology used to implement the objectives given in Chapter I.

Documentation

The documentation that needed to be developed for PCRSIM before any enhancements could be made were the programmatic flow of the source files and executable modules and a maintenance manual. These two documents were needed to describe the autodependencies of the individual programs. These documents provide the knowledge necessary to successfully build the five executable modules that comprise PCRSIM. Autodependency of a program refers to the individual subroutines and functions that are required to successfully compile and link the program into an executable module.

Programmatic Flow. To develop the programmatic flow of the source files and executable modules the code was examined and the function calls were traced. By tracing the flow of the function calls it is possible to determine the calling sequence of each program. Appendix B contains the block diagrams that describe the flow of the executable modules and the source files that comprise each of the executable files.

Maintenance Manual. The maintenance manual, contained in Appendix B, was built from the programmatic flows. Once

the programmatic flows were determined it was possible to compile and link the source files to build the executable modules. All five of the executable modules were successfully compiled and linked, however REGRESS.EXE had a runtime error when tested. A more detailed examination of the source files yielded an exhaustive list of the individual functions in each of the separate source files.

Review of the CLIPPER™, a database maintenance system, documentation indicated that the C (programming language that the majority of PCRSN was written in) files comprising MASTER.EXE needed to be compiled using the MicroSoft Version 5.0 Optimizing Compiler. The MicroSoft compiler was required to ensure compatibility with the CLIPPER™ programs. Appendix B contains the options required to compile the source files and the batch files used for linking the files into the executable modules.

Graphics Capability

A limitation of PCRSN was that it did not provide a capability to print the standardized residual versus Y-hat, residuals versus Y-hat, and the standardized residuals versus rankits. PCRSN assumes a Color Graphics Adapter (CGA) monitor and sets the videomode to CGA graphics prior to writing the individual pixels (locations on the screen) on the monitor to display the plots.

Researching the literature on C graphics programming yielded an algorithm by Stevens that prints the contents of the screen on an Enhanced Color Graphics Adapter (EGA) monitor (Stevens,1988:377-379). The algorithm was modified for the decreased, although more universal capability of CGA monitors. The modified algorithm is as follows:

1. Set printer for appropriate line spacing.
2. Send 12 linefeeds to the printer to center the output.
3. Read pixel from the (0,0) location to determine the background color.
4. Loop through the rows on the screen printing the pixels that are different from the background color.

The above algorithm assumes an Epson compatible printer, due to the control characters sent to the printer to perform the line spacing and linefeeds. If any key is pressed during the printing the print is aborted.

The user interface was modified to query if a plot should be sent to the printer. After a plot is displayed on the screen the user enters a keystroke and is queried as to whether the plot should be (1) sent to the printer or (2) exit. If the response is '1' then the plot is redrawn and Stevens' algorithm is invoked. Figures 1 through 3 are samples of the printed graphical displays.

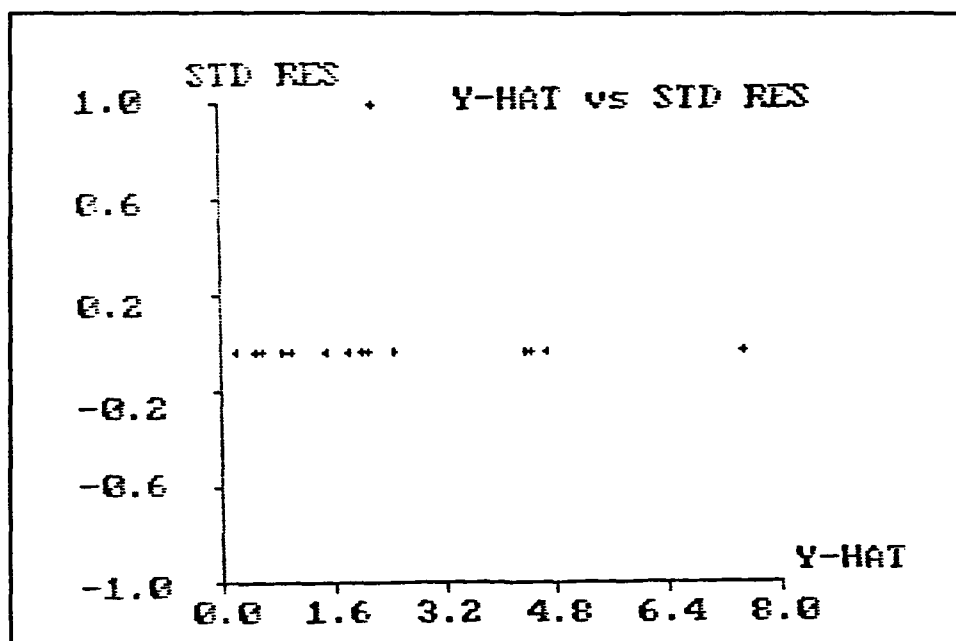


Figure 1. Standardized Residuals vs. Y-hat

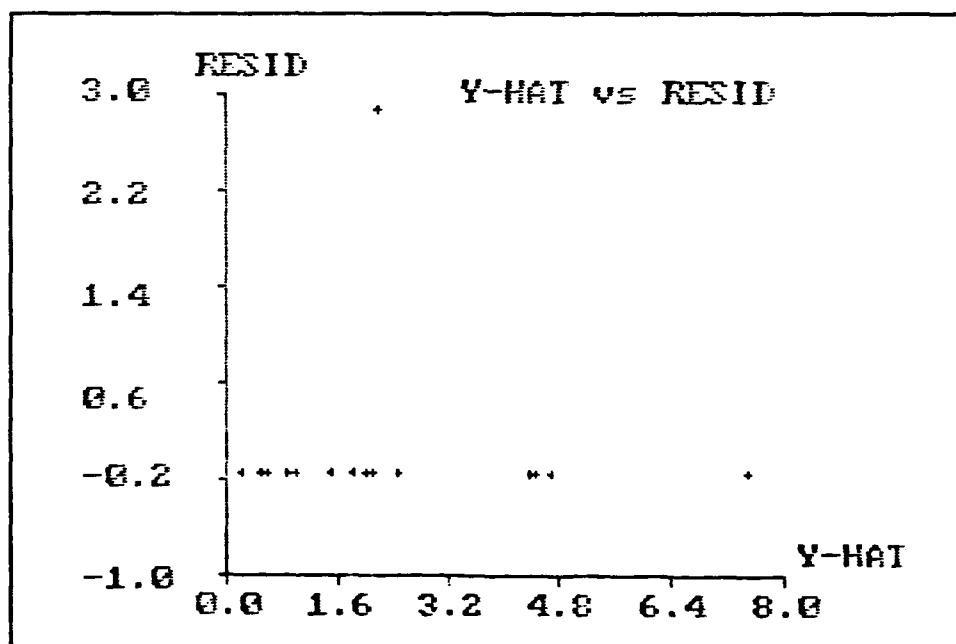


Figure 2. Residuals vs. Y-hat

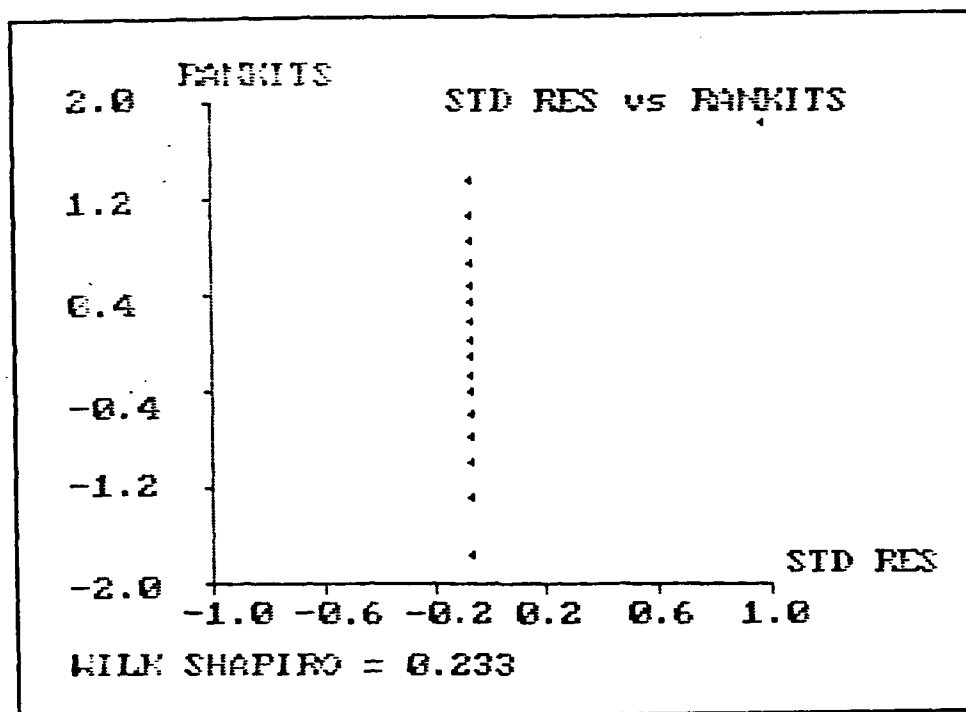


Figure 3. Standardized Residuals vs. Rankits

Variable Decoding

PCRSIM uses the encoded form of the factors. The encoded variables have values of -1 and 1 for two-level designs; and -1, 0, and 1 for three-level designs in the design matrix. The variables are encoded using the following equation

$$X_i = \frac{Z_i - M_i}{S_i} \quad (3)$$

where

Z_i = setting of i^{th} factor

M_i = midpoint of setting range of i^{th} variable

S_i = $1/2$ (range of i^{th} factor)

X_i = encoded value of i^{th} factor (Box and Draper, 1987:20)

Equation (3) standardizes the factor settings by mapping the Low/High factor settings of the original scale, Z_i , to -1 and 1 in the encoded variable, X_i .

For the purposes of clarification, define the regression equation as

$$\hat{Y} = \beta_0 + \beta_1 X_1 + \dots + \beta_{12} X_1 X_2 + \dots + \beta_{123} X_1 X_2 X_3 + \dots \quad (4)$$

where

β_0 = the intercept term
 β_i = coefficient of linear term X_i
 β_{ij} = coefficient of interaction term $X_i X_j$
 β_{ijk} = coefficient of interaction term $X_i X_j X_k$.

Define the decoded regression equation as

$$\hat{Y}_D = B_0 + B_1 Z_1 + \dots + B_{12} Z_1 Z_2 + \dots + B_{123} Z_1 Z_2 Z_3 + \dots \quad (5)$$

where

B_0 = the decoded intercept term
 B_i = decoded coefficient of linear term Z_i
 B_{ij} = decoded coefficient of interaction term $Z_i Z_j$
 B_{ijk} = decoded coefficient of interaction term $Z_i Z_j Z_k$.

Decoding the regression equation coefficients back to the original scale requires a linear transformation, substituting Equation (3) for the X_i 's in Equation (4).

The transformation can be accomplished by first setting $B_0 = \beta_0$ and then updating the decoded coefficients for each term of Equation (4) using Equations (6) through (19). When terms of higher order are involved, such as two- and three-way interactions, all terms of lower order are updated as well.

For linear terms, X_i update the coefficients in the following manner:

$$B_0 = B_0 - \frac{\beta_i M_i}{S_i} \quad (6)$$

$$B_i = B_i + \frac{\beta_i}{S_i} \quad (7)$$

For interaction terms, $X_i X_j$ update the coefficients in the following manner:

$$B_0 = B_0 + \frac{\beta_{ij} M_i M_j}{S_i S_j} \quad (8)$$

$$B_i = B_i - \frac{\beta_{ij} M_j}{S_i S_j} \quad (9)$$

$$B_j = B_j - \frac{\beta_{ij} M_i}{S_i S_j} \quad (10)$$

$$B_{ij} = B_{ij} + \frac{\beta_{ij}}{S_i S_j} \quad (11)$$

For interaction terms, $X_i X_j X_k$ update the coefficients in the following manner:

$$B_0 = B_0 - \frac{\beta_{ijk} M_i M_j M_k}{S_i S_j S_k} \quad (12)$$

$$B_i = B_i + \frac{\beta_{ijk} M_j M_k}{S_i S_j S_k} \quad (13)$$

$$B_j = B_j + \frac{\beta_{ijk} M_i M_k}{S_i S_j S_k} \quad (14)$$

$$B_k = B_k + \frac{\beta_{ijk} M_i M_j}{S_i S_j S_k} \quad (15)$$

$$B_{ij} = B_{ij} - \frac{\beta_{ijk} M_k}{S_i S_j S_k} \quad (16)$$

$$B_{ik} = B_{ik} - \frac{\beta_{ijk} M_j}{S_i S_j S_k} \quad (17)$$

$$B_{jk} = B_{jk} - \frac{\beta_{ijk} M_i}{S_i S_j S_k} \quad (18)$$

$$B_{ijk} = B_{ijk} + \frac{\beta_{ijk}}{S_i S_j S_k} \quad (19)$$

IV. Implementation of Methodology

The purpose of this chapter is to provide the details of implementing the methodology discussed in Chapter III.

Software Environment

PCRSM was developed using Turbo C++ and CLIPPER™. Additional details concerning version numbers and compatibility issues are covered in Appendix B.

Software Engineering as Applicable to PCRSM

Evaluation of the way that software engineering had been applied to PCRSM indicated deficiencies in the documentation. This lack of documentation presented difficulties in being able to successfully rebuild PCRSM from the source files.

Development of the programmatic flow and maintenance manuals proceeded as outlined in Chapter III with minor modification to the original software. For example, operating under the Turbo C++ environment required splitting Regout1.C into two separate source files due to a "lack of memory" error during compilation. As a result Regout1.C which was the program driver for the regression output section (REGOUT.EXE) of PCRSM was separated into the following two files. Regout1.C is the driver for the regression output, reading the input file (REGRESS.OUT)

written by REGRESS.EXE and controlling the execution of the Aptness Assessment portion of the output. Regout1a.C is called from Regout1.C for execution of the Model Results portion of the output.

After successfully compiling all source files, tracing the individual files determined the autodependencies of these files. This resulted in the error-free linking of the compiled object code into executable modules.

The programmatic flows and maintenance manual were developed as a direct result of the above efforts. The programmatic flows are diagrams depicting the flow of the executable modules and the program flow within each executable module. The maintenance manual contains the programmatic flows and a description of each source file as well as the required compile and link options. During this process it was discovered that the CLIPPERTM interface with C required MicroSoft C Version 5.0 Optimizing Compiler be used to build the C object files that are part of MASTER.EXE for compatibility reasons. See Appendix B for further details.

Graphics Capability

Implementing Stevens' modified algorithm provided the capability to print the graphical output of the Aptness Assessment portion of PCRSN. The REGOUT executable module of PCRSN, which controls the regression output was modified to include the graphic print capability.

REGOUT as developed by Leeper and Meidt was comprised of five C source files: Regout1.C (now Regout1.C and Regout1a.C), Regout2.C, Regout3.C, Regout4.C, and Regout5.C (see Appendix B). The functions that provide the graphical output are contained in Regout2.C. This program was modified to query the user for a response indicating whether to print the graph or not. If the user desired to print the graph it was redisplayed on the screen and Stevens' modified algorithm was called to send the contents of the screen to the printer.

Variable Decoding

Before executing the algorithm to decode the regression equation the program must determine if the user wants to decode the coefficients. This was accomplished by modifying the Model Results menu in REGOUT1A.C to provide a "Decode Coefficients" option for the user to select.

Assumptions. Three assumptions were required to implement the algorithm to decode the regression equation. The first was that there are less than 10 factors in the model. The second was that interaction terms of higher than third order are ignored. The third was that none of the factors were removed from the factor file by the user.

The first assumption enables the software to determine the order of the coefficients associated with the β 's. Because the coefficient labels are stored in the same array

as the estimates they are real numbers (floats). This poses a problem in determining whether a factor is an interaction term or a high linear term. For example, determining whether '12' is factor 12 or interaction term X_1X_2 . Limiting the number of variables for decoding to less than 10 allows the following algorithm to be used.

```

If number of factors < 10 then
  If label < 10 then linear term  $X_{label}$ 
  else
    i = label/10
    j = label - i
    if j <= number of factors then two-way
      interaction ij
    else
      k = i
      test = i/10
      j = i - (test * 10)
      i = test
      if i <= number of factors then three-
        way interaction ijk
      endif
    endif
  endif
else
  error message "more than 9 variables"
endif

```

The second assumption was made because of the complex nature of higher order interaction terms. These terms are almost impossible to interpret and the algebra involved in the linear transformation becomes extremely involved. To accommodate higher order interaction terms would require several special cases to be implemented in the software.

The third assumption was made because all the factors in the model are required to calculate the values needed to perform the linear transformation.

Implementation. If the user selects the "Decode Coefficients" option the program Regout1a.C calls the function Decode, contained in Regout6.C, passing the β coefficient estimates. Decode queries the user for the factor file name. The factor file, containing the number of factors in the model and the Low/High factor settings for each factor, was created by the user under the Experimental Design portion of PCRS. The midpoints (Midpt_i) and $1/2$ the ranges (S_i) are calculated from the Low/High setting values.

Once the constants (Midpt and S) used in the transformations are calculated Decode determines the location of the intercept term, β_0 and stores it in an array called UNCODED. If β_0 did not exist one was created. Decode then steps through the β array three times. First to update the linear terms, second to locate and update the two-way interaction terms by calling function twoway, and third to locate and update the three-way interaction terms by calling function threeway. Figure 4 contains a high level flowchart of function Decode.

Function twoway implements the algorithm to update the intercept, linear and interaction term as defined in Chapter III Equations (4), (5), and (6). The only modification was to check if the linear terms existed and if not to create them. This was required to allow for possible variable

selection by the user in the Regression Inputs portion of PCRS_M.

Function threeway is similar to function twoway except it uses Equations (7), (8), (9), and (10) to update the intercept, linear, two-way and three-way interaction terms.

After all calculations are made Decode displays the decoded regression coefficients on the screen and queries the user to determine whether to send the output to a file, the printer, or to exit. This portion has been implemented in the same manner as the other seven output tables of Model Results. See Appendix D for the source listing.

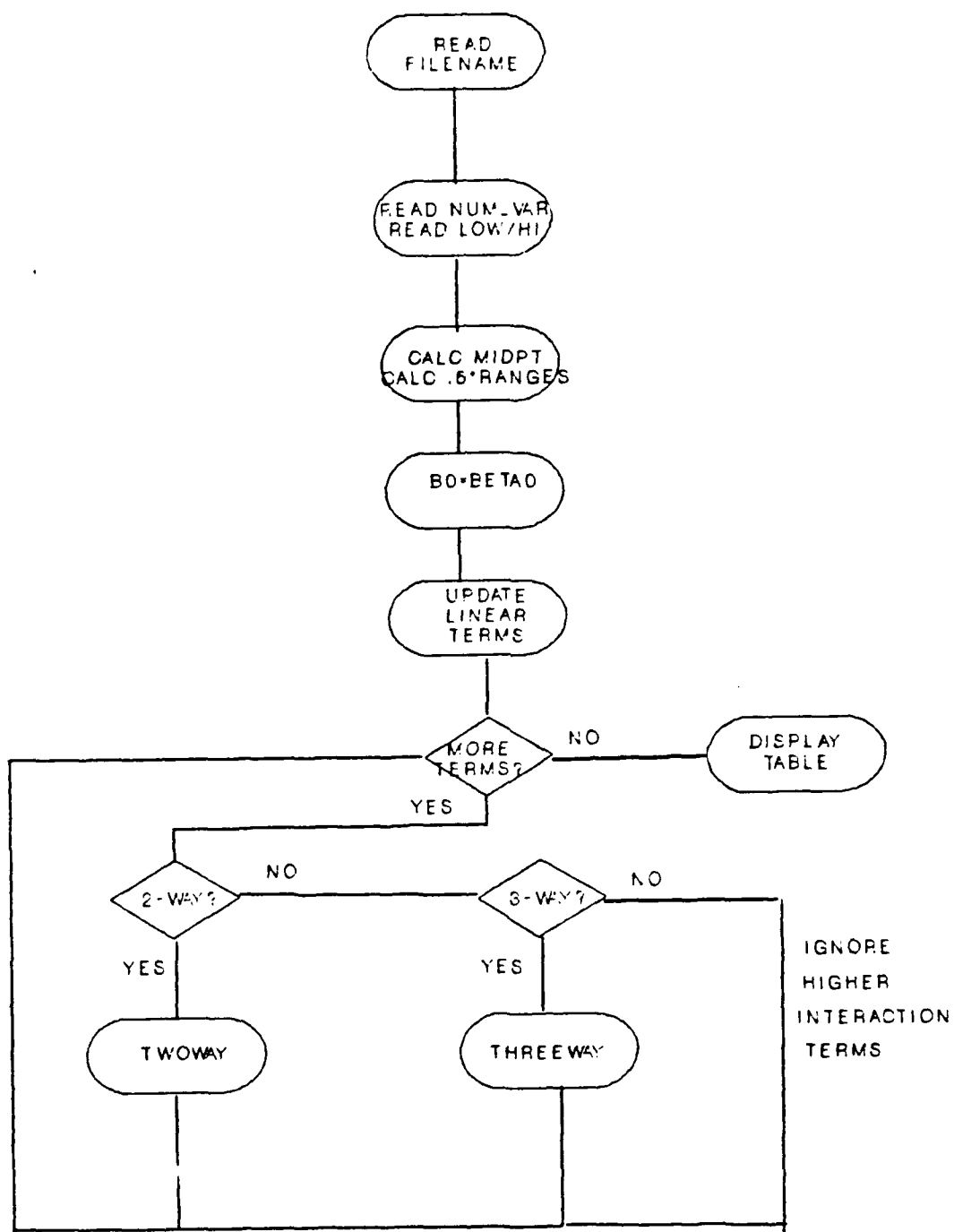


FIGURE 4. DECODE FLOWCHART

V. VERIFICATION AND VALIDATION

This chapter discusses the test cases that were run to perform verification and validation of the modifications made to PCRSIM. To test printing of the residual plots each graphic plot was printed. The three different types of graphs under the Aptness Assessment portion of the regression output can be found in Figures 1, 2, and 3 of Chapter III. The coefficient decoding software required much more extensive testing.

Graphics Printing

As mentioned above the capability to print the graphical displays was demonstrated in Figures 1 through 3 in Chapter III. Additional testing involved testing the print capability on different types of printers.

The graphical displays were printed on the Panasonic KX-1124 printer and an ALPS printer. In both cases the printed graph was a duplicate of the screen display.

Variable Decoding

Verification of the variable decoding was accomplished by applying Equations (6) through (19) of Chapter III to the individual test cases. This verified that the equations had been coded properly in the software. Validation was accomplished by evaluating both the regression equation and

the decoded regression equation of a particular model for given factor levels to ensure that both equations yielded the same predicted value.

The data use to generate the test cases extracted from Box and Draper is summarized in Table 1. The six cases that were run are listed in Table 2. Appendix C contains the Coefficient and Decoded Coefficient Tables generated for each case. Appendix C also contains the decoded regression coefficients calculated from the equations. The Decoded Coefficients table displays the decoded regression coefficients with five decimal place accuracy. This accuracy allows for more accurate estimation of the predicted values and is supported by the software. The regression coefficients are stored with higher than the three digit precision displayed on the output. The output displays are limited to three decimal digits to provide enough room for all the information in the various tables.

TABLE 1
TEST DATA

VARIABLE	LOW	HIGH
Length of specimen (mm)	250	350
Amplitude of load cycle (mm)	8	10
Load (g)	40	50

(Box and Draper, 1987:107)

TABLE 2
SUMMARIZATION OF TEST CASES

CASE 1	FULL FACTORIAL DESIGN
CASE 2	CASE 1 WITH INTERACTION TERMS REMOVED
CASE 3	CASE 1 WITH LINEAR TERM, X2 REMOVED
CASE 4	CASE 1 WITH INTERACTION TERM, X1X2 REMOVED
CASE 5	CASE 1 WITH ALL LINEAR TERMS REMOVED
CASE 6	3 X 3 X 3 FULL FACTORIAL DESIGN

The decoded regression coefficients calculated by PCRSIM and those calculated by the equations differed slightly. This difference was due to the fact that PCRSIM takes advantage of the higher precision of the regression coefficients in the software while the input into the equations only had three digit precision (Coefficient Table output). This difference was acceptable because the predicted values using the decoded regression equation from PCRSIM were within required accuracy (see Table 3).

The predicted value for each test case was determined using a different combination of the factor settings. In case 1 all factors were set high; in case 2 all factors were set low; in case 3 factor 1 was set low, and factors 2 and 3 were set high; in case 4 factor 1 was set high, and factors 2 and 3 were set low; in case 5 factors 1 and 2 were set low and factor 3 was set high; and in case 6 factor 1 was set high, factor 2 was set low, and factor 3 was set to the median level.

TABLE 3
COMPARISON OF PREDICTED VALUES

TEST CASE	ENCODED PREDICTED	DECODED PREDICTED	DIFFERENCE
CASE 1	2.56	2.56	0.0
CASE 2	2.84	2.84	0.0
CASE 3	2.245	2.245	0.0
CASE 4	3.285	3.285	0.0
CASE 5	2.695	2.725	0.03
CASE 6	3.377	3.924	0.547

The difference in the predicted value for Case 5 and Case 6 can attributed to the fact that the predicted values were calculated from coefficient estimates of differing precision. The estimates for the encoded values displayed in the Coefficient Table are only displayed with three digit precision. The estimates for the decoded values displayed in the Decoded Coefficient table are displayed with five digit precision.

This difference in display precision did not present a noticeable difference in the first four test cases because the cases were not as complicated. In Case 5 all linear terms were deleted and had to be reintroduced by the decoding calculation, and Case 6 was a quadratic model.

Summary

The capability to print the graphical displays was demonstrated in Chapter III and tested on different types of printers to test the compatibility of the software.

The decoding of the regression equation back to the original variable input scale has been extensively tested as discussed above. The testing did not reveal any inaccuracies in the developed algorithm or software implementation of the algorithm.

VI. CONCLUSIONS AND RECOMMENDATIONS

This chapter discusses the lessons learned from this research effort and makes recommendations for follow-on research.

Conclusions

The primary conclusion drawn from this research effort was that rapid prototyping as a system development technique, while producing an effective tool for the user does not allow for additional maintenance of the system unless appropriate documentation exists. Lack of documentation increases the time required to learn the system because additional time must be spent learning the programmatic flow of the system. Appropriate documentation alleviates the time spent on programmatic flow and allows an individual to start learning the code itself. Additional development efforts to PCRS, having overcome the documentation issue, were relatively straightforward.

Recommendations

The following is a list of recommendations for follow-on research. The recommendations are listed under the related executable module.

Master.

1. Rewrite MASTER.EXE source files entirely in C.

This would remove the CLIPPER™ files and reduce the size of the executable modules. Deleting CLIPPER™ also eliminates the requirement to compile the C files using the MicroSoft Version 5.0 Optimizing Compiler.

2. Modify the Box-Behnken design section to provide the user with the option of not orthogonalizing the design matrix. Performing this orthogonalization changes the factor settings slightly which can have a direct impact on the system being modeled.

3. Implement an algorithm to estimate the appropriate power transformation of the predictor variables (Box and Draper, 1987:296).

4. Add optimization of the response surface using canonical analysis. This involves determining the order of the terms for the regression coefficients (as done in DECODE), building the proper real symmetric, and calculating the eigenvalues. After the optimal values are determined a confidence region for the values must be determined. The optimization should work for both the encoded and the decoded regression equation. If applied to the decoded regression equation DECODE needs to be modified to write the decoded coefficients to a file to be read by the

optimization software. See Appendix E for background information on optimization.

Regout. Modify the Lack-of-Fit ANOVA table to separate the sums of squares by linear and pure quadratic elements.

Regress.

1. Determine cause of the run-time error (floating point error) when compiled and linked using Turbo C++.
2. Implement an algorithm to perform the calculations from Box and Behnken's original paper (Box and Behnken, 1960). The calculations determine the regression coefficients without performing the inversion of the matrix, $(X'X)$, where X is the Box-Behnken design matrix.

Summary

In summary this research effort achieved its objective to develop documentation for PCRSM and provide additional capabilities. Additional enhancements to PCRSM will provide the response surface methodology community with a valuable tool.

APPENDIX A: PCRSN USER'S MANUAL

INTRODUCTION

This user's manual was written by Leeper and Meidt (Leeper and Meidt, 1990: 73-74). It has been updated to reflect the enhancements made under this research effort, the printing of graphical information and decoding the regression coefficients.

PCRSN is a personal computer based decision support system for response surface methodology (RSM). The package is designed to help someone familiar with RSM perform RSM analysis on a PC.

PCRSN conducts group screening, factor screening, creates the coded experimental design matrices, creates raw data matrices in the original variable settings, and performs least squares regression on the design matrix and responses.

PCRSN contains six two and three level design types, including full factorials, fractional factorials, Plackett-Burman designs, central composite designs, and Box-Behnken designs. In all, there are 60 experimental designs for 2 to 39 factors.

The regression package performs model variable selection, performs transformations on the responses, provides a graphical model aptness assessment, and generates

a variety of model results. The model results include two ANOVA tables, a coefficient table, a decoded coefficient table, the variance-covariance matrix, the correlation matrix, and the design matrix. The user's guide is broken into three sections. The first section discusses the overall system operation and getting started. The second section describes the experimental design process. The third section describes the regression package.

SECTION 1: PCRSN OVERALL OPERATION

PCRSN was written for use on an IBM compatible PC. It uses five executable files. They are PCRSN.EXE, MASTER.EXE, REGRESS.EXE, REGOUT.EXE, and CLIPRAW.EXE. If you have them, you are ready to begin. As a side note, PCRSN requires much of a personal computer's 640k RAM memory to operate. Thus, you may have to remove Terminate and Stay Resident (TSR) programs (if you have any) before running PCRSN.

Type PCRSN to begin the program. The first screen should welcome you to PCRSN Version 2.0 and provide points of contact for comments and questions about the program.

The second screen provides the choices of conducting experimental design, running the regression package, or exiting. Highlight the choice you want by moving the up and down arrow keys and press <ENTER>.

SECTION 2: EXPERIMENTAL DESIGN PROCESS

Choosing EXPERIMENTAL DESIGN AND REGRESSION INPUTS from the main menu begins the experimental design process. In it you can conduct group screening, factor screening, create the coded experimental design matrices, create raw data matrices in the original variable settings, select the variables in the design matrix to regress, and perform transformations on the responses.

A. F1: THE HELP FUNCTION. Pressing the F1 key from the RSM Master Menu provides help. It includes definitions of key RSM terms and concepts as well as specific PCRSN options.

B. Esc: ESCAPE. When stuck in an undesirable situation, use the escape key to back out of the immediate process.

C. F10: MASTER, HOOK BOOK, NOTEPAD. The F10 key provides three options: Master, Hook book, and Notepad. Each option can be chosen by cursor selection or typing the first letter of the word.

The master option allows you to return to the RSM Master menu from any other menu.

The hook book is intended to be used as a method to capture system development and improvement thoughts. If an idea is generated at any time while in the analysis process, it can be immediately captured in the hook book without

disturbing the analysis itself. The hook book entries can be later reviewed, edited, or deleted.

The intent of the Notepad is to capture any thought, analysis related or not, for later review, edit, or deletion.

D. EXPERIMENTAL DESIGN. The experimental design choice generates three options: design choices, factor settings, and raw data matrix.

D1. Design Choices. Design choices determines and creates the experimental design based on the number of factors of interest, number of desired center points, number of desired design replications, and whether it's group or factor screening.

PCRSN contains six two and three level design types, including full factorials, fractional factorials, Plackett-Burman designs, central composite designs, and Box-Behnken designs. In all, there are 60 experimental designs for 2 to 39 factors.

When asked for the number of factors, if you want a two level design, enter the number of factors. The next screen will list the available two level design options. "RES" stands for resolution of the design and "RUNS" is the number of design points in the design. If you want a three level design instead, enter a 0 for the number of factors and the next screen will give you the three level design

options. You will then be asked a second time for the number of factors.

The Group screening option is only available with two level designs and will automatically generate the smallest (i.e., least number of runs) Resolution III design based on the number of groups.

After the design is created, it can be saved. An extension .DES (for DESign matrix) will be added to the filename. Even if it isn't saved, the design matrix will remain the current design available for further use.

D2. Factor Settings. The uncoded, or original, values of the factors can be entered, edited, or saved in the factor settings menu.

The data entry option reviews the experimental design to determine the number of factors. Enter the uncoded low and high values of the variables. If it's a three level design, the medium value will be calculated as the arithmetic mean of the high and low.

The variables and the variable settings can also be saved. An extension .FAC (for FACtor settings) will be added to the filename.

If you want to remove variables, choose the factor selection option. Select or highlight the variables you want to keep. If variables are removed, a new .FAC file can be created and saved.

To generate the Decoded Coefficient Table this file must be created. Because the method of decoding will create lower order terms when transforming interaction terms variables must not be removed from the .FAC file.

D3. Raw Data Matrix. The raw data matrix contains the variables and the uncoded settings of the variables required to generate the design points in the experimental design. It corresponds to the coded design matrix. To create the raw data matrix, retrieve the experimental design matrix and corresponding factor settings file. The program will automatically create the raw data matrix. If saved, the extension .RAW will be added.

Since the .RAW file is an ASCII file, it can easily be edited and potentially used to directly to run your model.

E. REGRESSION INPUTS. The regression inputs menu readies the design matrix and responses for the regression package. After entering the design matrix and responses, exit out to the PCRSN main menu to run the regression model.

E1. Design Matrix Input. The regression model needs to know which design matrix to use. If an experimental design was created previously in the same session, it does not need to be retrieved again. It will be used by default.

E2. Variable Selection. The experimental designs generated by PCRSN automatically create all possible

interaction terms, up to as many terms as rows in the design matrix. However, you may desire a more parsimonious model based on the results of a previous regression run or prior knowledge. The variable selection option allows you to choose the terms to leave in the model.

You can save the new design matrix if you want. But even if you don't save it, the regression model will use the new model unless a new design matrix is retrieved through the design matrix input option.

E3. Input Responses. The responses, or outputs, corresponding to the design matrix can be entered at the keyboard or read in from an ASCII file. However, before the responses can be entered at the keyboard, a design matrix must be retrieved so the total number of responses is known.

The format of the ASCII file must be as follows. The first entry on the first line must be the total number of responses. The responses should follow, one per line.

REMEMBER TO INPUT THE RESPONSES IN THE ORDER
CORRESPONDING TO THE DESIGN MATRIX SETTINGS!

E4. Response Transformations. Regression analysis sometimes indicates the responses should be transformed. PCRSM includes nine commonly used transformations. To perform a transformation on the responses, simply select the option. The original and transformed responses will appear on the screen. The

transformed responses do not need to be saved; the regression model will find the transformed data. If you want the responses in original values, select the original values option to delete the transformation.

SECTION 3: REGRESSION

The RUN REGRESSION AND OUTPUT RESULTS option performs the regression calculations and generates tabular and graphical displays of the regression statistics. The output contains aptness and model information. The tables can be sent to a file or printer.

A. APTNESS ASSESSMENT. The aptness assessment option provides graphical information on the standard regression model assumptions (i.e., normality, constant variance).

The first option contains a scatterplot of the standardized residuals versus the predicted value of the response, \hat{Y} . The second option contains a similar scatterplot containing the residuals versus \hat{Y} .

Option three plots the ordered standardized residuals versus rankits. The rankits are the expected value of an observation assuming a standard normal distribution. The plot also includes the Wilk-Shapiro test statistic for normality.

The graphical information can be sent to a local printer by selecting 1) SEND TO PRINTER. If any key is pressed during printing before the Aptness menu is

redisplayed the printing will be aborted. The software assumes an Epson compatible printer.

B. MODEL RESULTS. PCRSM provides eight tables, including include two ANOVA tables, a coefficient table, a decoded coefficient table, the variance-covariance matrix, the correlation matrix, and the design matrix.

B1. ANOVA Table. The Analysis of Variance Table displays the overall and individual sum of squares, F tests, and R^2 statistics. The individual sum of squares are sorted from largest to smallest. Thus, at a quick glance, the variables with the largest explanatory power can be identified.

B2. Coefficient Table. This table displays the coefficient values, standard errors, variances, student t statistics, and P-values. The coefficients are sorted by P-value from most significant variable to least significant.

B3. Decoded Coefficient Table. This table displays the decoded regression coefficient values. The decoded coefficients are displayed in the order presented in the design matrix unless a variable was removed from the model in which case it is displayed at the end of the list. Even if a variable has been removed from the model if a higher order term exists it will reintroduce the lower term. The software assumes less than 10 variables in the model and ignores greater than three-way interaction terms.

B4. Variance-Covariance Matrix. The variances of the variables are contained on the diagonal. If the design matrix is orthogonal, the off-diagonal elements will equal zero.

B5. Correlation Matrix. Since each variable is perfectly correlated with itself, the diagonal elements will equal one. If the design is orthogonal, the off-diagonal elements will equal zero.

B6. Lack of Fit ANOVA Table. This table is only available when at least one design point is replicated. It provides the same information as the ANOVA Table, except it replaces the extra sum of squares with lack of fit information. It also includes the lack of fit F statistic and P-value.

B7. Design Matrix. The Design Matrix contains the coded factor settings, the corresponding responses (Y), and the values of the predicted responses (Y-hat).

B8. $(X'X)^{-1}$ Matrix. The $(X'X)^{-1}$ matrix is used in the least squares regression calculations.

Appendix B: PCRSB MAINTENANCE MANUAL

PCRSB is comprised of five executable modules PCRSB, MASTER, CLIPRAW, REGRESS, and REGOUT. PCRSB ensures that the necessary files are available and is the driver for the main menu. MASTER provides the interface with the user for selection of the appropriate design and setting the input responses. In addition, MASTER calls CLIPRAW for the creation of the raw data matrix when required. REGRESS and REGOUT comprise the regression aspects of PCRSB; REGRESS performs the linear regression and REGOUT provides the output capabilities for the system. The following figure shows the relationship of the five modules.

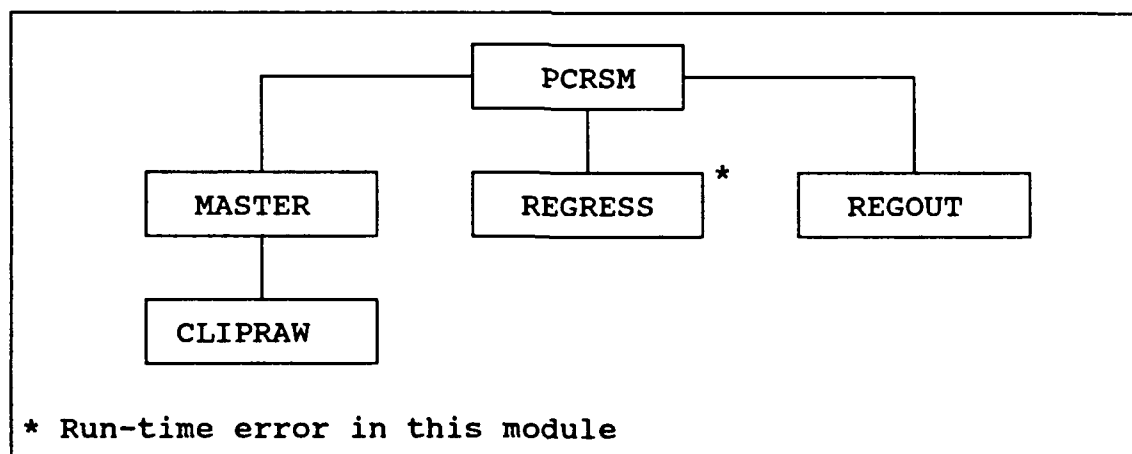


Figure 5. Executable Module Flowchart

Development Language

PCRSM was written in C and Clipper. Only MASTER contains source files that were written in Clipper to accommodate the menu scheme. The C source files were originally developed using Turbo C with follow on efforts under Borland Turbo C ++ Version 1.01 with the exception of the C files in MASTER which required MicroSoft C 5.0 for compatibility with Clipper. All of the Turbo C files are fully compatible with Turbo C ++.

Programmatic Flow

Figure 6 shows the individual source files that comprise each of the five executable modules.

Module Descriptions

This section contains a brief description of the source files in each of the executable modules.

PCRSM. PCRSM.C ensures that the required executable files exist: MASTER, REGRESS, and REGOUT. In addition, it displays the information screen that contains the version number and points of contacts for the system.

MASTER. MASTER is comprised of 11 Clipper files and five C files.

MASTER.PRG. This Clipper program provides the interface for the experimental design and regression input part of PCRSM.

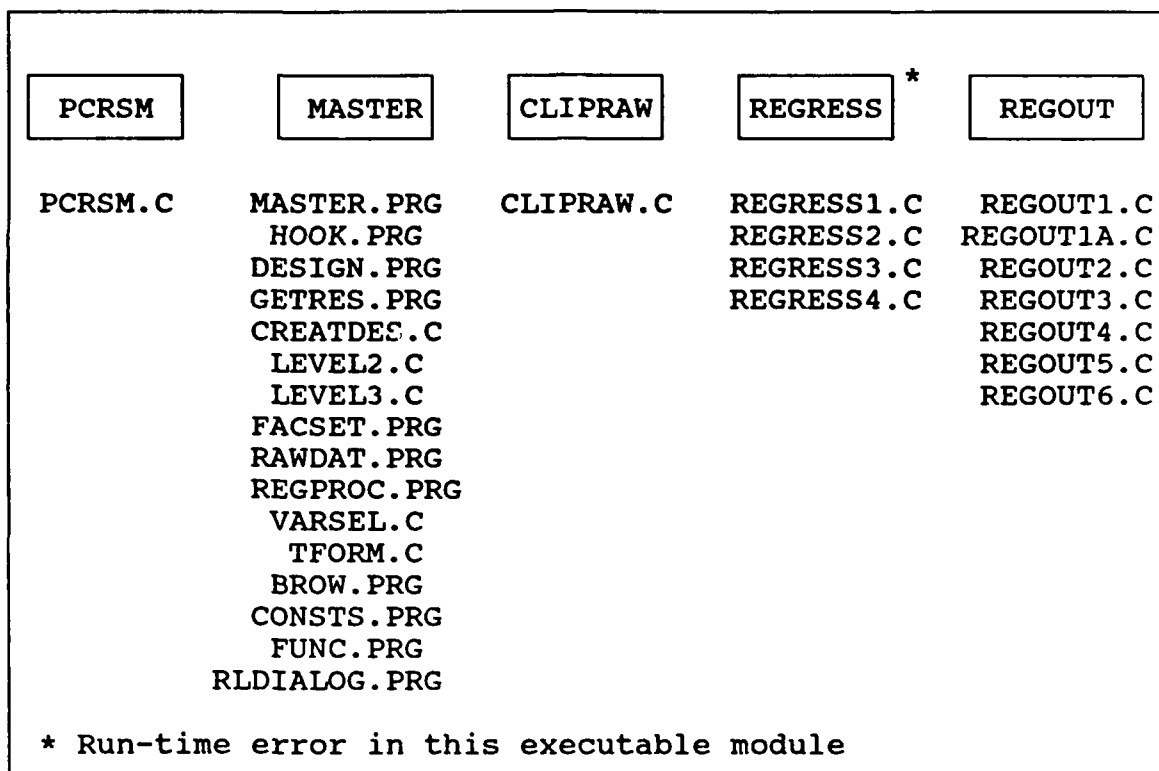


Figure 6. Programmatic Flow

HOOK.PRG. This Clipper file provides the capability to access the "hookbook" from anywhere in MASTER for the user to write notes on the system.

DESIGN.PRG. This Clipper file determines the type of experimental design, to include the resolution and number of replications and center points.

GETRES.PRG. This Clipper file determines the resolution and number of runs for the desired design.

CREATDES.C. This C file is "a function for Clipper to determine which design program to execute based on level and pass data" (Leeper and Meidt, 1990:157).

LEVEL2.C. This C file creates the level 2 design matrices, full factorial and Plackett-Burman designs.

LEVEL3.C. This C file creates the level 3 design matrices, central composite and Box-Behnken designs.

FACSET.PRG. This Clipper file provides the interface for entry of the high and low settings of the individual factors.

RAWDAT.PRG. This Clipper file provides the interface for creation of the raw data matrix and calls the CLIPRAW executable.

REGPROC.PRG. This Clipper file provides the interface for variable selection and transformation.

VARSEL.C. This C file performs the variable reduction of the design as indicated by the user.

TFORM.C. This C file performs transformation of the response variable as indicated by the user and writes the transformed data to a response file.

BROW.PRG. This Clipper file provides menu highlighting and control capabilities to MASTER.

CONSTS.PRG. This Clipper file contains the constants used in MASTER for menu and screen manipulation.

FUNC.PRG. This Clipper file determines filenames for files to be saved and retrieves appropriate files from disk.

RLDIALOG.PRG. This Clipper file is a modified Clipper system file to perform menubox operations.

CLIPRAW. CLIPRAW is comprised of one C file, CLIPRAW, that is a function for Clipper to write a file containing the raw data matrix.

REGRESS. REGRESS is comprised of four C files. REGRESS.EXE has been successfully compiled and linked in the Turbo C++ development environment, however, when PCRSM is executed with the rebuilt module a run-time error is encountered.

REGRESS1.C. This C file reads regression inputs from the file exp.des created by MASTER and performs least squares regression.

REGRESS2.C. This C file contains the following functions to support linear regression. Xpxinvf, to calculate the $(X'X)^{-1}$; betaf, to calculate the β parameters; yhatf, to calculate the y-hat statistics; ssrf, to calculate the uncorrected SSR; sstof, to calculate the uncorrected SSTO; matmulmm, to multiply an $(m \times n)$ matrix by an $(n \times m)$ matrix; matmulnn, to multiply an $(n \times 1)$ matrix by an $(1 \times n)$ matrix; matmul11, to multiply an $(1 \times n)$ matrix by an $(n \times 1)$ matrix; matmulm1, to multiply an $(m \times n)$ matrix by an $(n \times 1)$ matrix; matmul1m, to multiply an $(1 \times n)$ matrix by an $(n \times m)$ matrix; transpos, to transpose an $(m \times n)$ matrix;

transpos1, to transpose an (1 x m) matrix; and transpos2, to transpose an (m x 1) matrix.

REGRESS3.C. This C file contains the following functions to support linear regression. Rankits, to calculate the rankits; gammln, used in the calculation of the p-values; betai, "incomplete beta function used to generate T and Z values" (Leeper and Meidt, 1990:292); fvalue, "returns the p value for fstat [F statistic] and the two degrees of freedom" (Leeper and Meidt, 1990:293); xby, to calculate the orthogonal extra sums of squares; and extrassr, to calculate the non-orthogonal extra sums of squares.

REGRESS4.C. This C file contains the following functions to support linear regression. Invert, to invert an (n x n) matrix; factor, supports invert; and subst, a "substitution algorithm to support invert" (Leeper and Meidt, 1990:299).

REGOUT. REGOUT is comprised of seven C files.

REGOUT1.C. This C file provides the first regression output menu to the user and contains the functions to assess model aptness.

REGOUT1A.C. This C file provides the interface with the user for model results and calls the functions to perform the necessary calculations.

REGOUT2.C. This C file contains the following functions to support the linear regression output. Matmulmm, multiplies an (m x n) matrix by an (n x m) matrix; wilk, to calculate the Wilk-Shapiro statistic; plot, to plot the scatterplots to the screen; gridx, prints and labels the x-axis on the screen; gridy, prints and labels the y-axis; line, prints a line on the screen; mempoint, writes a point to the screen; mode, sets the video mode of the screen; gotoxy, moves the cursor to the position (x,y) on the screen; palette, sets the color palette for graphics; status, determines the status of the printer; read_Pixel, reads the pixel value from the screen; getmode, retrieves the video mode; printscr, sends the screen contents to the printer.

REGOUT3.C. This C file contains the following functions to support the linear regression output. Corr, writes the correlation matrix to the screen; corrprrt, sends the correlation matrix to the printer; corrrmat, calculates the correlation matrix.

REGOUT4.C. This C file contains the following functions to support the linear regression output. Covar, writes the variance-covariance matrix to the screen; covarprrt, sends the variance-covariance matrix to the printer; covrmat, calculates the variance-covariance matrix.

REGOUT5.C. This C file contains the following functions to support the linear regression output.

Xpxinvout, writes the $(X'X)^{-1}$ matrix to the screen; and
xpxinvprt, sends the $(X'X)^{-1}$ matrix to the printer.

REGOUT6.C. This C file contains the functions called by REGOUT1A.C to perform the decoding of the regression coefficients. Function decode queries the user for the file containing the factor setting levels and reads this file. Decode loops through the BETA array to determine what the label for each coefficient is and calculates the decoded linear, two-way and three-way interaction coefficients.

Compiling

This section discusses compiling the source files and the compile options required.

PCRS.M. The source file is compiled using the large memory model and the emulation package for floating point calls.

MASTER. All of the C files in MASTER must be compiled using the MicroSoft C 5.0 Optimizing Compiler to be compatible with Clipper. The command is:

```
CL /c /AL /Zl /Oalt /FPa /Gs filename
```

where

CL	- compile linker command
/c	- compile only
/Al	- large memory model

/Zl - suppresses default library selection and produces smaller object files
/Oalt - optimization with relaxed alias checking, enabled loop optimization and favoring execution speed over size
/FPa - floating point alternate math package
/Gs - remove calls to stack-checking routines.
(CLIPPER, 1988:11-5)

The following is a listing of the batch file MASTER.CLP that is used to compile the CLIPPER files into object file master.obj.

```
master
design
facset
rawdat
regproc
hook
```

The execution command is: CLIP @master. The remaining five CLIPPER programs are compiled individual with the command: CLIP filename.

CLIPRAW. The source file is compiled using the large memory model and the emulation package for floating point calls.

REGRESS. The source files are compiled using the large memory model and the emulation package for floating point calls.

REGOUT. The source files are compiled using the large memory model and the emulation package for floating point calls.

Linking

Turbo C++ provides the ability to maintain project files. Project files contain a list of all the source files to be compiled and linked into one executable module. A project file keeps track of the last compile and link so that only those portions of a project that need to be recompiled are compiled. If the compile defaults are set appropriately the project manager links in the necessary libraries.

All of the executable modules in PCRSIM, except MASTER can be managed using project files. The project files are created by inserting filenames into a project. The project files of PCRSIM contain the files listed in Figure 5 for each executable module.

MASTER. The following batch file, MASTER.LNK, is used to link the object modules that comprise MASTER.EXE.

```
OUTPUT MASTER.EXE
FI MASTER,GETRES
FI
A:\OBJECT\TFORM,A:\OBJECT\VARSEL,A:\OBJECT\CREATDES,A:\
OBJECT\LEVEL2
FI A:\OBJECT\LEVEL3
FI CONSTS,FUNC,BROW,RLDIALOG MIXCASE
LIB \CLIPPER\CLIPPER,\CLIPPER\EXTEND,\LIB\LLIBCA
STACK 8000
```

The batch program is executed using the PLINK86 linker provided with CLIPPER. The execution command is: PLINK86 @master. The LLIBCA library is a MicroSoft Version 5.0 Runtime Library built for the large memory model with graphics and the alternate floating point routines.

APPENDIX C: TEST CASES RESULTS

CASE 1: 2^3 FULL FACTORIAL DESIGN

COEFFICIENT TABLE					
WARNING ! STATISTICAL TESTS ARE INVALID. SSE = 0 WITH df = 0. THEY HAVE BEEN ARBITRARILY SET TO ONE.					
VARIABLE	VALUE	STD ERROR	VARIANCE	STUDENT-T	P-VALUE
X0	2.745	0.354	0.125	7.764	0.0817
X1	0.375	0.354	0.125	1.061	0.4819
X2	-0.295	0.354	0.125	-0.834	0.5567
X3	-0.175	0.354	0.125	-0.495	0.7070
X123	-0.040	0.354	0.125	-0.113	0.9282
X23	-0.020	0.354	0.125	-0.057	0.9640
X12	-0.015	0.354	0.125	-0.042	0.9730
X13	-0.015	0.354	0.125	-0.042	0.9730

DECODED COEFFICIENTS	
WARNING ! ADDITIONAL TERMS MAY HAVE BEEN INTRODUCED DURING DECODING	
VARIABLE	VALUE
Z0	20.92499
Z1	-0.05190
Z2	-2.18500
Z3	-0.41300
Z12	0.00690
Z13	0.00138
Z23	0.04400
Z123	-0.00016

TABLE 4
RESULTS FROM TEST CASE 1

	CODED	DECODED	CALCULATED
β_0	2.745	20.92499	22.545
β_1	0.375	-0.05190	-0.052
β_2	-0.295	-2.18500	-2.185
β_3	-0.175	-0.41300	-0.413
β_{12}	-0.015	0.00690	0.007
β_{13}	-0.015	0.00138	0.001
β_{23}	-0.020	0.04400	0.044
β_{123}	-0.040	-0.00016	-1.6E-4

CASE 2: FULL FACTORIAL WITH INTERACTION TERMS REMOVED

COEFFICIENT TABLE					
VARIABLE	VALUE	STD ERROR	VARIANCE	STUDENT-T	P-VALUE
X0	2.745	0.025	0.001	110.926	0.0000
X1	0.375	0.025	0.001	15.154	0.0001
X2	-0.295	0.025	0.001	-11.921	0.0003
X3	-0.175	0.025	0.001	-7.072	0.0021

DECODED COEFFICIENTS

WARNING ! ADDITIONAL TERMS MAY HAVE BEEN INTRODUCED
DURING DECODING

VARIABLE	VALUE
Z0	4.72500
Z1	0.00750
Z2	-0.29500
Z3	-0.03500

TABLE 5

RESULTS FROM TEST CASE 2

	CODED	DECODED	CALCULATED
β_0	2.745	4.72500	4.725
β_1	0.375	0.00750	.008
β_2	-0.295	-0.29500	-.295
β_3	-0.175	-0.03500	-.035

CASE 3: LINEAR TERM, X2, REMOVED FROM FULL FACTORIAL DESIGN

COEFFICIENT TABLE					
VARIABLE	VALUE	STD ERROR	VARIANCE	STUDENT-T	P-VALUE
X0	2.745	0.295	0.087	9.305	0.0682
X1	0.375	0.295	0.087	1.271	0.4249
X3	-0.175	0.295	0.087	-0.593	0.6587
X123	-0.040	0.295	0.087	-0.136	0.9141
X23	-0.020	0.295	0.087	-0.068	0.9568
X12	-0.015	0.295	0.087	-0.051	0.9676
X13	-0.015	0.295	0.087	-0.051	0.9676

DECODED COEFFICIENTS	
WARNING ! ADDITIONAL TERMS MAY HAVE BEEN INTRODUCED DURING DECODING	
VARIABLE	VALUE
Z0	18.26999
Z1	-0.05190
Z3	-0.41300
Z12	0.00690
Z13	0.00138
Z23	0.04400
Z123	-0.00016
Z2	-1.89000

TABLE 6
RESULTS FROM TEST CASE 3

	CODED	DECODED	CALCULATED
β_0	2.745	18.26999	20.700
β_1	0.375	-0.05190	-0.055
β_2		-1.89000	-2.340
β_3	-0.175	-0.41300	-0.413
β_{12}	-0.015	0.00690	0.007
β_{13}	-0.015	0.00138	0.001
β_{23}	-0.020	0.04400	0.044
β_{123}	-0.040	-0.00016	-1.6E-4

CASE 4: INTERACTION TERM, X12, REMOVED FROM FULL FACTORIAL DESIGN

COEFFICIENT TABLE					
VARIABLE	VALUE	STD ERROR	VARIANCE	STUDENT-T	P-VALUE
X0	2.745	0.015	0.000	183.167	0.0035
X1	0.375	0.015	0.000	25.023	0.0255
X2	-0.295	0.015	0.000	-19.685	0.0324
X3	-0.175	0.015	0.000	-11.677	0.0545
X123	-0.040	0.015	0.000	-2.669	0.2285
X23	-0.020	0.015	0.000	-1.335	0.4099
X13	-0.015	0.015	0.000	-1.001	0.5004

DECODED COEFFICIENTS

WARNING ! ADDITIONAL TERMS MAY HAVE BEEN INTRODUCED
DURING DECODING

VARIABLE	VALUE
----------	-------

Z0	21.73499
Z1	-0.05460
Z2	-2.27500
Z3	-0.41300
Z13	0.00138
Z23	0.04400
Z123	-0.00016
Z12	0.00720

TABLE 7

RESULTS FROM TEST CASE 4

	CODED	DECODED	CALCULATED
β_0	2.745	21.73499	20.205
β_1	0.375	-0.05460	-0.055
β_2	-0.295	-2.27500	-2.340
β_3	-0.175	-0.41300	-0.413
β_{12}		0.00720	0.007
β_{13}	-0.015	0.00138	0.001
β_{23}	-0.020	0.04400	0.044
β_{123}	-0.040	-0.00016	-1.6E-4

CASE 5: ALL LINEAR TERMS REMOVED FROM FULL FACTORIAL DESIGN

COEFFICIENT TABLE					
VARIABLE	VALUE	STD ERROR	VARIANCE	STUDENT-T	P-VALUE
X0	2.745	0.293	0.086	9.355	0.0026
X123	-0.040	0.293	0.086	-0.136	0.9000
X23	-0.020	0.293	0.086	-0.068	0.9499
X12	-0.015	0.293	0.086	-0.051	0.9624
X13	-0.015	0.293	0.086	-0.051	0.9624

DECODED COEFFICIENTS	
WARNING ! ADDITIONAL TERMS MAY HAVE BEEN INTRODUCED DURING DECODING	
VARIABLE	VALUE
Z0	18.94499
Z12	0.00690
Z13	0.00138
Z23	0.04400
Z123	-0.00016
Z1	-0.05940
Z2	-1.89000
Z3	-0.37800

TABLE 8
RESULTS FROM TEST CASE 5

	CODED	DECODED	CALCULATED
β_0	2.745	18.94499	20.565
β_1		-0.05940	-0.059
β_2		-1.89000	-2.250
β_3		-0.37800	-0.450
β_{12}	-0.015	0.00690	0.007
β_{13}	-0.015	0.00138	0.001
β_{23}	-0.020	0.04400	0.044
β_{123}	-0.040	-0.00016	-1.6E-4

CASE 6: 3 X 3 X 3 FULL FACTORIAL DESIGN

COEFFICIENT TABLE					
VARIABLE	VALUE	STD ERROR	VARIANCE	STUDENT-T	P-VALUE
X0	2.751	0.016	0.000	173.518	0.0000
X1	0.362	0.019	0.000	18.628	0.0000
X2	-0.274	0.019	0.000	-14.107	0.0000
X3	-0.171	0.019	0.000	-8.813	0.0000
X123	-0.040	0.029	0.001	-1.373	0.1914
X13	-0.029	0.024	0.001	-1.227	0.2404
X11	-0.037	0.034	0.001	-1.107	0.2872
X33	-0.029	0.034	0.001	-0.859	0.4051
X12	-0.014	0.024	0.001	-0.596	0.5612
X23	-0.010	0.024	0.001	-0.421	0.6808
X22	0.013	0.034	0.001	0.380	0.7100

DECODED COEFFICIENTS

WARNING ! ADDITIONAL TERMS MAY HAVE BEEN INTRODUCED
DURING DECODING

VARIABLE	VALUE
----------	-------

Z0	18.23073
Z1	-0.04083
Z2	-2.48889
Z3	-0.30922
Z11	-0.00001
Z22	0.01278
Z33	-0.00116
Z12	0.00692
Z13	0.00132
Z23	0.04600
Z123	-0.00016

TABLE 9

RESULTS FROM TEST CASE 6

	CODED	DECODED	CALCULATED
β_0	2.751	18.23073	16.130
β_1	0.362	-0.04083	-0.045
β_2	-0.274	-2.48889	-2.377
β_3	-0.171	-0.30922	-0.361
β_{11}	-0.037	-0.00001	-2.96E-5
β_{22}	0.013	0.01278	0.026
β_{33}	-0.029	-0.00116	-0.002
β_{12}	-0.014	0.00692	0.007
β_{13}	-0.029	0.00132	0.001
β_{23}	-0.010	0.04600	0.046
β_{123}	-0.040	-0.00016	-1.6E-4

APPENDIX D: SOURCE LISTING

PCRSM

The following source file comprises the PCRSM executable module. It provides the initial information screen and verifies that the required executable files exist. The program also passes control to appropriate executable module based on user selection.

PCRSM.C.

```
/* RSM.C -- DRIVER MENU FOR RSM IN EXPERIMENTAL DESIGN AND
REGRESSION */
/* THIS PROGRAM NEEDS ANSI.SYS LOADED IN THE CONFIG.SYS
FILE */
/* AS IN "DEVICE=ANSI.SYS" */
/* MODIFICATION HISTORY
   11 DEC 90  INCLUDE THE DIR.H FILES AND SET VERSION
              NUMBER BY CAPT ROUILLARD
   15 JAN 91  UPDATE VERSION NUMBER BY CAPT ROUILLARD
              */

# include <string.h>
# include <stdlib.h>
# include <dos.h>
# include <dir.h>
# include <io.h>
# include <stdio.h>
#define TRUE 1
#define NUM 3
#define CLEAR    "\x1B[2J"      /* clears screen */
#define ERASE    "\x1B[K"      /* erases line from cursor
position */
#define NORMAL   "\x1B[0m"      /* normal attribute */
#define BOLD     "\x1B[1m"
#define BLINK    "\x1B[5m"
#define REVERSE  "\x1B[7m"      /* reverse video attribute */
#define LEFTCOL  "\x1B[%dC"
#define ROWCOL   "\x1B[%d;%df"
#define HOME     "\x1B[11;20f" /* the starting position of
the menu */
#define BOTTOM    "\x1B[24;1f"  /* the starting position of
the message line */
```

```

#define U_ARROW 72          /* the extended code for up
arrow */
#define D_ARROW 80          /* the extended code for down
arrow */
#define RETURN 13          /* the code for carriage
return */
#define BLOCK "\xDB"
#define UHALF "\xDF"
#define LHALF "\xDC"
#define ULCORNER "\xC9"
#define URCORNER "\xBB"
#define LLCORNER "\xC8"
#define LRCORNER "\xBC"
#define HDOUBLE "\xCD"
#define VDOUBLE "\xBA"
int return_val;
char mas_filename[12] = "master.exe";
char reg_filename[12] = "regress.exe";
char out_filename[12] = "regout.exe";
/* struct find_t exe_file; */
struct ffbk ffbk;

main()
{

static char *items[NUM] =
    { "EXPERIMENTAL DESIGN AND REGRESSION INPUTS",
      "      RUN REGRESSION AND OUTPUT RESULTS      ",
      "                                QUIT                                ", };
static char *message[NUM] =
    { "SELECT TO CREATE EXPERIMENTAL DESIGNS AND CREATE
REGRESSION INPUTS",
      "SELECT TO RUN REGRESSION",
      "SELECT TO QUIT TO DOS", };

int curpos;
int code;

/* write initial program description screen */
fscreen();

curpos=0;
while (TRUE) {
    printf(CLEAR);
    display_box();
    display(items,message,NUM,curpos);
    code = getcode();
    switch (code)
    {
    case U_ARROW:

```

```

        if(curpos > 0) --curpos; break;
    case D_ARROW:
        if(curpos < NUM-1) ++curpos; break;
    case RETURN:
        printf(CLEAR);
        action(curpos); break;
    }
}}

/* display box around menu */
display_box()
{
    int i;
    printf(NORMAL);
    printf(ROWCOL,8,26);
    printf("<=====| RSM MENU |=====>");
    printf(ROWCOL,10,19);
    printf(ULCORNER); for (i=1; i<=41; i++) printf(HDOUBLE);
    printf(URCORNER);
    printf(ROWCOL,14,19);
    printf(LLCORNER); for (i=1; i<=41; i++) printf(HDOUBLE);
    printf(LRCORNER);
    for (i=1; i<=3; i++) {
        printf(ROWCOL,10+i,19);
        printf(VDOUBLE);
    }
    for (i=1; i<=3; i++) {
        printf(ROWCOL,10+i,61);
        printf(VDOUBLE);
    }
}

/* displays menu */
display(arr,msg,size,pos)
char *arr[];
char *msg[];
int size,pos;
{
    int j,result,center_pos;
    int r,c;
    for (j=0; j<size; j++) {
        r = 11+j; c = 20;
        printf(ROWCOL,r,c); /* position cursor on screen */
        if (j == pos)
            printf(REVERSE); /* print item in reverse */
        printf("%s\n",*(arr+j)); /* print the item */
        printf(NORMAL); /* print in normal mode */
    }
    printf(BOTTOM); /* cursor to lower left */
    printf(ERASE);
}

```

```

    result = strlen(*(msg+pos));
    center_pos = (80 - result)/2 ;
    printf("\x1B[24;%df",center_pos); /* center message */
    printf("%s",*(msg+pos)); /* print message associated with
menu item */
}

/* gets the keyboard input */
getcode()
{
    int key,exit_while = 0;
    while (exit_while != 1) {
        key = getch();
        if (key == 0) {
            key = getch() ; /* get the extend code */
            exit_while = 1;
        }
        else if (key == 13) exit_while = 1 ; /* allow the
carriage return */
    }
    return( key ) ; /* return key code value */
}

/* performs action based on cursor position */
action(pos)
int pos;
{
    switch(pos) {
        case 0:
            return_val = findfirst(mas_filename,&ffblk,0);
            if ( return_val == 0)
                return_val = system("master");
            else {
                printf(BOTTOM);
                printf(ERASE);
                printf("ERROR - THE FILE => %s <= DOES NOT EXIST IN
DIRECTORY <RETURN>"
,mas_filename);
                getch();
            }
            break;
        case 1:
            return_val = findfirst(reg_filename,&ffblk,0);
            if ( return_val == 0) {
                /* DELETE THE REGRESS.OUT FILE SO THAT IF AN ERROR
OCCURS */
                /* IN THE REGRESSION -- THE REGOUT PROGRAM WILL NOT RUN
*/
                return_val = remove("regress.out");
                return_val = system("regress");
            }
    }
}

```

```

    }
    else {
        printf(BOTTOM);
        printf(ERASE);
        printf("ERROR - THE FILE => %s <= DOES NOT EXIST IN
DIRECTORY <RETURN>"
,reg_filename);
        getch();
    }
    return_val = findfirst(out_filename,&ffblk,0);
    if ( return_val == 0) {
        if ( findfirst("regress.out",&ffblk,0) == 0)
            return_val = system("regout");
        else {
            printf(BOTTOM);
            printf(ERASE);
            printf("ERROR - THE REGRESSION DID NOT PRODUCE AN
OUTPUT FILE <RETURN>");
            getch();
        }
    }
    else {
        printf(BOTTOM);
        printf(ERASE);
        printf("ERROR - THE FILE => %s <= DOES NOT EXIST IN
DIRECTORY <RETURN>"
,out_filename);
        getch();
    }
    break;
case 2:
    printf(CLEAR);
    exit(0);
}
}
fscreen()
{
    int i;
    printf(CLEAR);
    printf(BOLD);

    /* print first row */
    printf(ROWCOL,1,22);
    printf(BLOCK UHALF UHALF UHALF BLOCK " ");
    printf(BLOCK UHALF UHALF UHALF BLOCK " " BLOCK);
    for (i=1; i<=3; i++) printf(UHALF);
    printf(LHALF " " BLOCK);
    for (i=1; i<=4; i++) printf(UHALF);
    printf(" " BLOCK BLOCK LHALF " " LHALF BLOCK BLOCK);

```

```

/* print the second row */
printf(ROWCOL,2,22);
printf(BLOCK " " BLOCK " " BLOCK);
printf(LEFTCOL,6);
printf(BLOCK " " BLOCK " ");
printf(BLOCK " " BLOCK " " BLOCK LHALF BLOCK " "
BLOCK);

/* print the third row */
printf(ROWCOL,3,22);
printf(BLOCK UHALF UHALF UHALF UHALF " " BLOCK " ");
printf(BLOCK UHALF BLOCK UHALF " ");
printf(UHALF BLOCK LHALF " " BLOCK " " UHALF);
printf(BLINK); printf("\x02"); printf(NORMAL); printf(BOLD);
printf(UHALF " " BLOCK);

/* print the fourth row */
printf(ROWCOL,4,22);
printf(BLOCK " " BLOCK " " BLOCK);
printf(" " UHALF BLOCK LHALF);
printf(LEFTCOL,5);
printf(UHALF BLOCK " " BLOCK " " BLOCK);

/* print the fifth row */
printf(ROWCOL,5,22);
printf(BLOCK " " BLOCK LHALF LHALF LHALF BLOCK " "
BLOCK);
printf(" " BLOCK " ");
for (i=1; i<=4; i++) printf(LHALF);
printf(BLOCK " " BLOCK " " BLOCK);
printf(NORMAL);

/* print the system name and authors */
printf(ROWCOL,7,25);
printf(ULCORNER);
for (i=1; i<=28; i++) printf(HDOUBLE);
printf(URCORNER);
printf(ROWCOL,8,25);
printf(VDOUBLE " DECISION SUPPORT SYSTEM " VDOUBLE);
printf(ROWCOL,9,25);
printf(VDOUBLE " FOR " VDOUBLE);
printf(ROWCOL,10,25);
printf(VDOUBLE "RESPONSE SURFACE METHODOLOGY" VDOUBLE);
printf(ROWCOL,11,25);
printf(VDOUBLE " VERSION 2.0 " VDOUBLE);
printf(ROWCOL,12,25);
printf(VDOUBLE " BY " VDOUBLE);
printf(ROWCOL,13,25);
printf(VDOUBLE " CAPT LAURIE M. ROUILLARD " VDOUBLE);
printf(ROWCOL,14,25);

```



```

printf(VDOUBLE "      CAPT DAVID M. LEEPER      " VDOUBLE);
printf(ROWCOL,15,25);
printf(VDOUBLE "      CAPT GREGORY J. MEIDT      " VDOUBLE);
printf(ROWCOL,16,25);
printf(LLCORNER);
for (i=1; i<=28; i++) printf(HDOUBLE);
printf(LRCORNER);

/* print the information line at bottom */
printf(ROWCOL,18,8);
printf("IF YOU HAVE ANY QUESTIONS OR COMMENTS ABOUT THIS
PROGRAM CONTACT:\n\n");
printf("      CAPT LAURIE M. ROUILLARD AFIT/ENS
      AV 785-3362\n");
printf("      CAPT DAVID M. LEEPER      PENTAGON/X0X1
      AV 225-1535\n");
printf("      CAPT GREGORY J. MEIDT      USAFA/DFM
      AV 259-4130\n");
printf("      MAJ KEN BAUER      AFIT/ENS
      AV 785-3362\n");
printf("      ----- PRESS ANY KEY TO CONTINUE
      ");
printf("-----");
getch();
}

```

MASTER

The following 16 source files comprise the MASTER executable module. Eleven of the files are CLIPPER™ source files, the remaining five are C source files.

MASTER.PRG

```
*
* MASTER.PRG
*
CLEAR
init_consts()    && initialize all of the program constants
SET KEY F10 TO menu_line
level = 2
num_rows = 0
num_groups = 0
des_filename = SPACE(12)
res_filename = SPACE(12)
exit_now = .F.
SET MESSAGE TO 24 CENTER
DO WHILE !exit_now
    BEGIN SEQUENCE
        * DRAW THE OUTSIDE OUTBOX AND FILL MIDDLE OF BOX
        @ 1,1,23,79 BOX sl_box + chr(177)
        PUBLIC loc_array[5]
        PRIVATE s_locwin[5]
        num_levels = 1
        * UPDATE THE LOCATION BOX
        loc_array[num_levels] = menu_pad("MASTER MENU",22)
        s_locwin[num_levels] = current_loc(num_levels)
        prompt_size = 20

        DO box_top_msg WITH 9,29,14,50,box1,"RSM",.F.,norm,bright
        @ 24,0 SAY SPACE(80)
        @ 10,30 PROMPT menu_pad("Experimental
Design",prompt_size);
            MESSAGE "SELECT TO EXECUTE EXPERIMENTAL DESIGN
OPTION"
        @ 11,30 PROMPT menu_pad("Regression Inputs",prompt_size);
            MESSAGE "SELECT TO CREATE OR ADJUST REGRESSION
INPUTS"
        @ 12,30 PROMPT menu_pad("Optimization",prompt_size);
            MESSAGE "OPTION CURRENTLY UNAVAILABLE"
        @ 13,30 PROMPT menu_pad("Quit",prompt_size);
            MESSAGE "SELECT TO QUIT TO THE DOS SYSTEM"
        choice = 1
        MENU TO choice
```

```

DO CASE
  CASE choice = 1
    * UPDATE LOCATION BOX
    num_levels = num_levels + 1
    loc_array[num_levels] = menu_pad("EXPERIMENTAL
DESIGN",22)
    s_locwin[num_levels] = current_loc(num_levels)

    exp_prompt_size = 26
    save_expwin = savescreen(14,26,19,53)
    exit_expdesign = .F.
    DO WHILE !exit_expdesign
      DO box_top_msg WITH 14,26,19,53,box1,"EXPERIMENTAL
DESIGN", ;
        .F.,norm,bright
        SET MESSAGE TO 24 CENTER
        @ 24,0 SAY SPACE(80)
        @ 15,27 PROMPT menu_pad("Design
Choices",exp_prompt_size);
        MESSAGE "SELECT TO CREATE THE LEVEL 2 OR 3
DESIGN MATRIX"
        @ 16,27 PROMPT menu_pad("Factor
Settings",exp_prompt_size);
        MESSAGE "SELECT TO INPUT THE LOW AND HIGH
FACTOR VALUES"
        @ 17,27 PROMPT menu_pad("Raw Data
Matrix",exp_prompt_size);
        MESSAGE "SELECT TO CREATE THE RAW DATA MATRIX"
        @ 18,27 PROMPT menu_pad("Exit",exp_prompt_size);
        MESSAGE "SELECT TO EXIT"
      choice = 1
      MENU TO choice
      DO CASE    && CASE STRUCTURE FOR EXPERIMENTAL DESIGN
        CASE choice = 1    && DESIGN CHOICES

          * UPDATE LOCATION BOX
          num_levels = num_levels + 1
          loc_array[num_levels] = menu_pad("DESIGN
CHOICES",22)
          s_locwin[num_levels] = current_loc(num_levels)

          DO p_design WITH level,num_rows,num_groups

          * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP
          restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
          num_levels = num_levels - 1

        CASE choice = 2    && FACTOR SETTINGS

```

```

        * UPDATE LOCATION BOX
        num_levels = num_levels + 1
        loc_array[num_levels] = menu_pad("FACTOR
SETTINGS",22)
        s_locwin[num_levels] = current_loc(num_levels)

        DO p_facset WITH level,num_rows

        * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP
restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
        num_levels = num_levels - 1

        CASE choice = 3          && RAW DATA MATRIX

        * UPDATE LOCATION BOX
        num_levels = num_levels + 1
        loc_array[num_levels] = menu_pad("RAW DATA
MATRIX",22)
        s_locwin[num_levels] = current_loc(num_levels)

        DO p_rawdat WITH num_groups

        * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP
restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
        num_levels = num_levels - 1

        CASE choice = 4          && END EXPERIMENTAL DESIGN
MENU
        exit_expdesign = .T.

        ENDCASE          && FOR EXPERIMENTAL DESIGN MENU OPTIONS
        ENDDO
        restscreen(14,26,19,53,save_expwin)

        * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP
restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
        num_levels = num_levels - 1

        CASE choice = 2
        * UPDATE LOCATION BOX
        num_levels = num_levels + 1
        loc_array[num_levels] = menu_pad("REGRESSION",22)
        s_locwin[num_levels] = current_loc(num_levels)

        reg_prompt_size = 26
        design_input   = .F.
        response_input = .F.
        exit_regdesign  = .F.

```

```

        save_regwin = savescreen(14,26,20,53)
        DO WHILE !exit_regdesign
        DO box_top_msg WITH 14,26,20,53,box1,"REGRESSION", ;
            .F.,norm,bright
            @ 24,0 SAY SPACE(80)
            @ 15,27 PROMPT menu_pad("Design Matrix
Input",reg_prompt_size);
                MESSAGE "SELECT TO RETRIEVE A DESIGN MATRIX"
            @ 16,27 PROMPT menu_pad("Variable
selection",reg_prompt_size);
                MESSAGE "SELECT TO REDUCE THE NUMBER OF
VARIABLES"
            @ 17,27 PROMPT menu_pad("Input
Responses",reg_prompt_size);
                MESSAGE "SELECT TO INPUT RESPONSE DATA"
            @ 18,27 PROMPT
menu_pad("Transformations",reg_prompt_size);
                MESSAGE "SELECT TO TRANSFORM THE RESPONSE
DATA"
            @ 19,27 PROMPT menu_pad("Exit",reg_prompt_size);
                MESSAGE "SELECT TO EXIT"
            choice = 1
            MENU TO choice
            DO CASE      && CASE STRUCTURE FOR REGRESSION
                CASE choice = 1      && DESIGN MATRIX INPUT

                    * UPDATE LOCATION BOX
                    num_levels = num_levels + 1
                    loc_array[num_levels] = menu_pad("DESIGN MATRIX
INPUT",22)
                    s_locwin[num_levels] = current_loc(num_levels)

                    @ 0,1 SAY "SELECT THE DESIGN FILE NAME <EXP.DES>
IS CURRENT"
                    des_filename = get_filename("*.DES")
                    @ 0,0 SAY SPACE(80)

                    IF !EMPTY(des_filename)
                        * make the input filename the current design file
                        COPY FILE &des_filename TO exp.des
                        design_input = .T.
                    ENDIF

                    * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP

restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
                    num_levels = num_levels - 1

                CASE choice = 2      && VARIABLE SELECTION

```

```

        * UPDATE LOCATION BOX
        num_levels = num_levels + 1
        loc_array[num_levels] = menu_pad("VARIABLE
SELECTION",22)
        s_locwin[num_levels] = current_loc(num_levels)

        IF design_input
            DO p_varsel WITH des_filename
        ELSE
            @ 0,1 SAY "ERROR -> NO DESIGN INPUT <RETURN>"
            wait_key = inkey(0)
            @ 0,0 SAY SPACE(80)
        ENDIF

        * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP
        restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
        num_levels = num_levels - 1

        CASE choice = 3          && RESPONSE INPUT

            * UPDATE LOCATION BOX
            num_levels = num_levels + 1
            loc_array[num_levels] = menu_pad("RESPONSE
INPUT",22)
            s_locwin[num_levels] = current_loc(num_levels)

            DO p_response
            IF !EMPTY(res_filename)
                response_input = .T.
            ENDIF

            * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP
            restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
            num_levels = num_levels - 1

            CASE choice = 4          && RESPONSE TRANSFORMATIONS

                * UPDATE LOCATION BOX
                num_levels = num_levels + 1
                loc_array[num_levels] =
menu_pad("TRANSFORMATIONS",22)
                s_locwin[num_levels] = current_loc(num_levels)

                IF response_input
                    DO p_transform
                ELSE
                    @ 0,1 SAY "ERROR -> NO RESPONSE FILE INPUT
<RETURN>"

```

```

        wait_key = inkey(0)
        @ 0,0 SAY SPACE(80)
    ENDIF

    * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP

restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
    num_levels = num_levels - 1

    CASE choice = 5      && END REGRESSION MENU
        exit_regdesign = .T.

    ENDCASE      && FOR REGRESSION MENU OPTIONS
    ENDDO
    restscreen(14,26,20,53,save_regwin)

    * RESTORE SCREEN FOR LOCATION BOX NEXT LEVEL UP
    restscreen(2,55,3+num_levels,78,s_locwin[num_levels])
    num_levels = num_levels - 1

    CASE choice = 3
    CASE choice = 4
        exit_now = .T.
    ENDCASE && FOR MASTER MENU OPTIONS
    END && END FOR SEQUENCE STRUCTURE
ENDDO
SET CURSOR ON
CLEAR
RETURN

PROCEDURE help
PARAMETERS proc,line,var
    SET KEY F1 TO
    SAVE SCREEN
    @ 0,0 SAY SPACE(80)
    @ 24,0 SAY SPACE(80)
    @ 0,1 SAY "USE THE ARROWS AND PAGE UP/DOWN TO MOVE INSIDE
THE HELP WINDOW"
    IF FILE("pcrsm.hlp")
        @ 1,1,23,79 BOX sl_box + chr(177)
        @ 8,10 SAY "HELP INFORMATION FOR SYSTEM -- PRESS ESC TO
LEAVE"
        @ 9,1,19,79 BOX SL_BOX
        MEMOEDIT(MEMOREAD("pcrsm.hlp"),10,2,18,78,.F.)
    ELSE
        @ 0,0 SAY SPACE(80)
        @ 0,1 SAY "ERROR -- HELP FILE -> PCRSM.HLP <- NOT
AVAILABLE <RETURN>"
        wait_key = inkey(0)
    ENDIF

```

```

    RESTORE SCREEN
    SET KEY F1 TO help
    RETURN

```

```

PROCEDURE menu_line
PARAMETERS proc,line,var
SET KEY F10 TO
SAVE SCREEN
@ 0,0 SAY SPACE(80)
@ 24,0 SAY SPACE(80)
@ 0,01 PROMPT "Master" ;
    MESSAGE "SELECT TO GET THE MASTER MENU"
@ 0,09 PROMPT "Hook book" ;
    MESSAGE "SELECT TO WRITE DOWN SYSTEM ENHANCEMENTS"
@ 0,30 PROMPT "Notepad" ;
    MESSAGE "SELECT TO WRITE DOWN ANY NOTES"
choice = 1
MENU TO choice
DO CASE
    CASE choice = 1
        IF var = "CHOICE"
            @ 0,0 SAY SPACE(80)
            SET KEY F10 TO menu_line
            CLEAR
            KEYBOARD CHR(ENTER)
            BREAK
        ELSE
            @ 0,1 SAY "ERROR -- ONLY FROM A MENU => PRESS ESC TO
GET TO MENU <RETURN>"
            wait_key = inkey(0)
            RESTORE SCREEN
        ENDIF
    CASE choice = 2
        @ 1,1,23,79 BOX sl_box + chr(177)
        DO p_hook WITH .T.
        RESTORE SCREEN
    CASE choice = 3
        @ 1,1,23,79 BOX sl_box + chr(177)
        DO p_hook WITH .F.
        RESTORE SCREEN
ENDCASE
@ 0,0 SAY SPACE(80)
SET KEY F10 TO menu_line
RETURN

```

```

PROCEDURE box_top_msg
PARAM t,l,b,r,box_str,str,intense,norm,bright
PRIVATE l_spaces,r_spaces,msg_len

```



```

@ t,l,b,r BOX box_str
msg_len= len(str)
l_spaces = int(((r-l-1)-msg_len)/2)
r_spaces = l_spaces
* adjust left side if not equal
IF r_spaces + l_spaces + msg_len != (r-l-1)
    l_spaces = l_spaces + 1
ENDIF
@ t,l SAY substr(box_str,1,1)
@ t,l+1 SAY replicate(substr(box_str,2,1),l_spaces)
IF intense
    SET COLOR TO &bright
    @t,l+1_spaces+1 SAY str
    SET COLOR TO &norm
ELSE
    @ t,l+1_spaces+1 SAY str
ENDIF
@ t,l+1_spaces + 1 + len(str) SAY ;
    replicate(substr(box_str,2,1),r_spaces)
@ t,r SAY substr(box_str,3,1)
RETURN

FUNCTION current_loc
PARAMETERS num_levels
PRIVATE save_locwin
save_locwin = savescreen(2,55,3+num_levels,78)
DO box_top_msg WITH 2,55,3+num_levels,78,box2,"CURRENT
LOCATION", ;
    .F.,norm,bright
FOR i = 1 to num_levels
    IF i = num_levels
        SET COLOR TO I
        @ 2+i,56 SAY loc_array[i]
        SET COLOR TO
    ELSE
        @ 2+i,56 SAY loc_array[i]
    ENDIF
NEXT
RETURN (save_locwin)

```

HOOK.PRG

```

*
* PROGRAM HOOK.PRG
*
PROCEDURE p_hook
PARAMETERS hook_flag
* single line box
*CLEAR

```

```

*SL_BOX = chr(218) + chr(196) + chr(191) + chr(179) +
chr(217) + ;
*      chr(196) + chr(192) + chr(179)
*hook_flag = .F.
IF hook_flag
    dbase_name = "hook"
    start_pos = 9
    db_message = "HOOK BOOK"
ELSE
    dbase_name = "notepad"
    start_pos = 30
    db_message = "NOTEPAD"
ENDIF
dbfilename = dbase_name + ".dbf"
IF !FILE(dbfilename)
    CREATE dummy
    APPEND BLANK
    REPLACE field_name WITH "date_time",;
        field_type WITH "C", ;
        field_len WITH 17, ;
        field_dec WITH 0
    APPEND BLANK
    REPLACE field_name WITH "descript",;
        field_type WITH "C", ;
        field_len WITH 40, ;
        field_dec WITH 0
    APPEND BLANK
    REPLACE field_name WITH "text",;
        field_type WITH "C", ;
        field_len WITH 700, ;
        field_dec WITH 0

    CREATE &dbfilename FROM dummy
    ERASE dummy.dbf
ENDIF
USE &dbase_name
@ 24,0 SAY SPACE(80)
@ 1,start_pos,4,start_pos+7 BOX SL_BOX
@ 2,start_pos+1 PROMPT "Add " ;
    MESSAGE "SELECT TO ADD " + db_message + "
ENTRY"
@ 3,start_pos+1 PROMPT "Review" ;
    MESSAGE "SELECT TO PREVIOUS " + db_message +
" ENTRIES"
hook_choice = 1
MENU TO hook_choice
DO CASE
    CASE hook_choice = 1
        @ 5,16 SAY "<CTRL>-W TO SAVE ENTRY -OR- ESC TO ABORT
ENTRY"

```

```

@ 7,29 SAY "----- " + db_message + " -----"
@ 8,1,18,79 BOX SL_BOX    && BOX FOR MENU FIELD
memo_string = MEMOEDIT(SPACE(700),9,2,17,78,.T.)
* ONLY SAVE RESULTS IF MEMO IS NOT BLANK BECAUSE OF
ABORT
IF !EMPTY(memo_string)
    description = SPACE(40)
    @ 21,2 SAY "INPUT A SHORT DESCRIPTION OF " + db_message
+ " ENTRY"
    @ 22,2 SAY "DESCRIPTION =>" GET description
    READ
    APPEND BLANK
    REPLACE date_time WITH DTOC(DATE()) + "|" + TIME()
    REPLACE descript WITH description
    REPLACE text WITH memo_string
ENDIF
@ 0,0 SAY SPACE(80)
@ 24,0 SAY SPACE(80)
CASE hook_choice = 2
    * ALWAYS START AT TOP OF DATA BASE
    last_record = RECCOUNT()
    GO TOP
    review_exit = .F.
    review_choice = 1
    @ 24,0 SAY SPACE(80)
    @ 7,29 SAY "----- " + db_message + " -----"
    @ 2,start_pos+8,6,start_pos+17 BOX SL_BOX    && BOX FOR
MENU SELECTIONS

    @ 8,1,18,79 BOX SL_BOX    && BOX FOR MENU FIELD
    @ 19,33 SAY " RECORD # 1  "
    @ 21,2 SAY "DATE & TIME => " + date_time
    @ 22,2 SAY "DESCRIPTION => " + descript + SPACE(40 -
LEN(descript))
    display_text()
    DO WHILE !review_exit
        @ 3,start_pos+9 PROMPT "Next      " ;
        MESSAGE "SELECT TO SEE NEXT " +
db_message + " ENTRY"
        @ 4,start_pos+9 PROMPT "Previous" ;
        MESSAGE "SELECT TO SEE PREVIOUS " +
db_message + " ENTRY"
        @ 5,start_pos+9 PROMPT "Exit      " ;
        MESSAGE "SELECT TO EXIT"
    MENU TO review_choice

    IF review_choice = 3    && exit
        review_exit = .T.
    ELSE
        error_msg = SPACE(14)

```

```

IF review_choice = 1
  IF RECNO() != last_record
    SKIP 1
  ELSE
    error_msg = "AT END OF FILE"
  ENDIF
ELSE
  IF RECNO() != 1
    SKIP -1
  ELSE
    error_msg = "AT TOP OF FILE"
  ENDIF
ENDIF
IF EMPTY(error_msg)
  rec_string = LTRIM(STR(RECNO()))
  @ 19,33 SAY " RECORD # " + rec_string +
SPACE(4-LEN(rec_string))
  @ 21,2 SAY "DATE & TIME => " + date_time
  @ 22,2 SAY "DESCRIPTION => " + descript + SPACE(40 -
LEN(descript))
  display_text()
ELSE
  @ 19,33 SAY error_msg
ENDIF
ENDIF
ENDDO
ENDCASE
RETURN

```

```

FUNCTION display_text
PRIVATE ret_val
  row = 9
  col = 2
  width = 77
  wrap = .F.
  FOR curr_line = 1 TO 9
    print_line = MEMOLINE(text,width,curr_line)
    @ row + curr_line-1,col SAY print_line
  NEXT
  ret_val = 0
RETURN ret_val

```

DESIGN.PRG

```

*
*  DESIGN.PRG
*
PROCEDURE p_design
PARAMETERS level,num_rows,num_groups

```

```

BEGIN SEQUENCE
    PRIVATE ar[20],desarr[8]
    IF FILE("exp.des")
        ERASE exp.des
    ENDIF
    num_center_pts = 0
    num_reps = 0
    num_rows = 0
    save_reswin = savescreen(14,10,20,42)
    SET KEY ESC TO proc_esc
    info_msg = "THE NUMBER OF FACTORS HELP DETERMINE
THE MODEL"
    @ 0,1 SAY info_msg
    @ 24,1 SAY "Input the number of FACTORS" + ;
        " [ENTER 0 FOR THREE-LEVEL MODELS] =>"
;
        GET num_rows VALID validate(1,info_msg)
    READ
    @ 0,0 SAY SPACE(80)
    @ 24,0 SAY SPACE(80)
    IF num_rows = 0    && THREE-LEVEL MODELS APPLY
        size = 3      && NUMBER OF ITEMS TO DISPLAY
        ar[1] = "FULL FACTORIAL"
        ar[2] = "CCD"
        ar[3] = "BOX-BEHNKEN"
        level = 3
        orthogonal = 0
        alpha = 0.00000
        @ 14,10 SAY "THREE - LEVEL MODELS" + SPACE(14)
        SET KEY ESC TO ESC
        sel = abrowse(ar,size,ENTER,15,10,20,42)
        IF sel = 0    && IF ESCAPE KEY PRESSED THEN EXIT
            restscren(14,10,20,42,save_reswin)
            RETURN
        ENDIF
        SET KEY ESC TO proc_esc
        num_rows = 2
        info_msg = "THIS IS THE ACTUAL NUMBER OF FACTORS
" + ;
            "FOR THE 3-LEVEL DESIGNS"
        @ 0,1 SAY info_msg
        @ 24,1 SAY "Input the number of FACTORS =>" ;
            GET num_rows VALID validate(2,info_msg)
        READ
        @ 0,0 SAY SPACE(80)
        @ 24,0 SAY SPACE(80)

        IF sel < 3    && OTHER THAN BOX-BEHNKEN DESIGN
            * MUST HAVE AT LEAST ONE CENTER POINT FOR CCD
OPTION

```

```

        IF sel = 2
            num_center_pts = 1
        ENDIF
        info_msg = "PROVIDES INFORMATION ABOUT " + ;
                    "THE MIDDLE OF THE DESIGN"
        @ 0,1 SAY info_msg
        @ 24,1 SAY "Input the number of CENTER POINTS
" + ;
                    "<RETURN FOR NONE> =>" GET
num_center_pts ;
                    VALID validate(3,info_msg)
        READ
        @ 0,0 SAY SPACE(80)
        @ 24,0 SAY SPACE(80)

        info_msg = "ONE REPLICATION --> REPLICATES THE
ENTIRE DESIGN"
        @ 0,1 SAY info_msg
        @ 24,1 SAY "Input the number of REPLICATIONS
" + ;
                    "<RETURN FOR NONE> =>" GET
num_reps ;
                    VALID validate(4,info_msg)
        READ
        @ 0,0 SAY SPACE(80)
        @ 24,0 SAY SPACE(80)

        IF sel = 2    && CCD DESIGN
            SET FIXED ON
            alpha = calc_alpha(num_rows,num_center_pts)
            SET FIXED OFF
            compare_alpha = alpha
            @ 0,1 SAY "ALPHA VALUE CONTROLS WHETHER OR
NOT " + ;
                    "DESIGN IS ORTHOGONAL"
            @ 24,1 SAY "If factors can be set to " + ;
                    "orthogonal ALPHA - <RETURN> " +
;
                    "else <VALUE> " GET alpha    PICT
"99.99999"

            READ
            @ ^,0 SAY SPACE(80)
            @ 24,0 SAY SPACE(80)
            IF alpha = compare_alpha
                orthogonal = 1
            ENDIF
        ENDIF
    ENDIF
*   BUILD THE 3-LEVEL DESIGN CREATION PARAMETER ARRAY

```

```

        desarr[1] = num_groups
        desarr[2] = INT(num_rows)
        desarr[3] = level
        desarr[4] = sel
        desarr[5] = orthogonal
        desarr[6] = alpha
        desarr[7] = INT(num_center_pts)
        desarr[8] = num_reps
    @ 0,0 SAY SPACE(80)
    @ 0,0 SAY ltrim(str(desarr[1])) + " | " +
ltrim(str(desarr[2])) + ;
    " | " + ltrim(str(desarr[3])) + " | " +
ltrim(str(desarr[4])) + ;
    " | " + ltrim(str(desarr[5])) + " | " +
ltrim(str(desarr[6])) + ;
    " | " + ltrim(str(desarr[7])) + " | " +
ltrim(str(desarr[8]))
    WAIT_K = INKEY(0)
    @ 0,0 SAY SPACE(80)

        ELSE      &&  TWO - LEVEL DESIGNS
        DECLARE res[4],runs[4]
        afill(res,0)
        afill(runs,0)
        num_elements = 1

* GET THE DESIGN'S RESOLUTION AND NUMBER OF RUNS BASED ON
LEVEL & # FACTORS
        DO p_getres WITH
level,num_rows,res,runs,num_elements

        ar[1] = "    GROUP SCREENING"  && ADD OPTION TO
LIST
        FOR i = 1 TO num_elements
            IF res[i] = 0
                ar[i+1] = "    FULL FACTORIAL    " +
STR(runs[i],3)
            ELSE
                ar[i+1] = SPACE(3) + STR(res[i],1) +
SPACE(16) ;
                                + STR(runs[i],3)
            ENDIF
        NEXT
        @ 14,10 SAY "    RES                      # RUNS" +
SPACE(18)
        IF num_rows >= 12
            @ 0,1 SAY "BECAUSE THE NUMBER OF FACTORS IS
GREATER THAN" ;
                                + " 11 - SUGGEST GROUP SCREENING"
        ENDIF

```

```

size = num_elements + 1
SET KEY ESC TO ESC
sel = abrowse(ar,size,ENTER,15,10,20,42)
  IF sel = 0    && IF ESCAPE KEY PRESSED THEN EXIT
    restscreen(14,10,20,42,save_reswin)
    RETURN
  ENDIF
SET KEY ESC TO proc_esc

@ 0,0 SAY SPACE(80)

resolution
* based on the "sel" value get the proper
* to pass to the routine to get the value
* check to see if group screening was selected

sel = 1
level = 2
num_groups = 0
IF sel = 1    && GROUP SCREENING
  num_groups = 1
  info_msg = "ASSISTS IN REDUCING THE FACTOR
SPACE"
  @ 0,1 SAY info_msg
  @ 24,1 SAY "Input the number of GROUPS " + ;
    "<RETURN FOR NONE> =>" GET

num_groups ;
      VALID validate(5,info_msg)
      READ
      @ 0,0 SAY SPACE(80)
      @ 24,0 SAY SPACE(80)
      ENDIF
      info_msg = "PROVIDES INFORMATION ABOUT " + ;
        "THE MIDDLE OF THE DESIGN"
      @ 0,1 SAY info_msg
      @ 24,1 SAY "Input the number of CENTER POINTS "
+ ;
      "<RETURN FOR NONE> =>" GET

num_center_pts ;
      VALID validate(3,info_msg)
      READ
      @ 0,0 SAY SPACE(80)
      @ 24,0 SAY SPACE(80)

      info_msg = "ONE REPLICATION --> REPLICATES THE
ENTIRE DESIGN"
      @ 0,1 SAY info_msg
      @ 24,1 SAY "Input the number of REPLICATIONS "
+ ;

```



```

                                "<RETURN FOR NONE> =>" GET num_reps
;
                                VALID validate(4,info_msg)
                                READ
                                @ 0,0 SAY SPACE(80)
                                @ 24,0 SAY SPACE(80)

                                * BUILD THE DESIGN CREATION PARAMETER ARRAY
                                desarr[1] = num_groups
                                desarr[2] = num_rows
                                desarr[3] = level
                                IF num_groups > 0
                                    * IF GROUP SCREENING THEN USE LOWEST
RESOLUTION DESIGN <3>
                                    DO CASE
                                        CASE num_groups = 2
                                            resolution = 0
                                        CASE num_groups = 4
                                            resolution = 4
                                        OTHERWISE
                                            resolution = 3
                                    ENDCASE
                                desarr[4] = resolution
                                ELSE
                                    * sel one less because group screening is
first option
                                    IF sel > 1
                                        desarr[4] = res[sel-1]
                                    ENDIF
                                ENDIF
                                desarr[5] = num_center_pts
                                desarr[6] = num_reps
                                @ 0,0 SAY SPACE(80)
                                @ 0,0 SAY ltrim(str(desarr[1])) + " | " +
ltrim(str(desarr[2])) + ;
                                " | " + ltrim(str(desarr[3])) + " | " +
ltrim(str(desarr[4])) + ;
                                " | " + ltrim(str(desarr[5])) + " | " +
ltrim(str(desarr[6]))
                                WAIT_K = INKEY(0)
                                @ 0,0 SAY SPACE(80)
                                ENDIF
                                SET CURSOR ON
                                * CALL THE C CODE FUNCTION TO BUILD THE DESIGN
                                SAVE SCREEN
                                CLEAR
                                *? "THIS IS WHERE THE C CODE TAKES OVER -- PRESS RETURN"
                                *wait_key = inkey(0)
                                cdesign(desarr)
                                RESTORE SCREEN

```

```

        restscreen(14,10,20,42,save_reswin)
        IF FILE("exp.des")
            overwrite = .F.
            prompt_msg = "PLEASE ENTER THE DESIGN MATRIX
FILENAME"
            des_file_name =
input_filename(".DES",@overwrite,prompt_msg)
            IF !EMPTY(des_file_name) .AND. overwrite
                COPY FILE exp.des TO &des_file_name
            ENDIF
        ENDIF
    END    && FOR END SEQUENCE
    SET KEY ESC TO ESC
    RETURN

PROCEDURE proc_esc
    @ 0,0 SAY SPACE(80)
    restscreen(14,10,20,42,save_reswin)
    BREAK
    RETURN

FUNCTION calc_alpha
PARAMETERS num_factors,center_pts
PRIVATE num_factors,center_pts,F,T,Q
    SET DECIMALS TO 5
    F = 2**num_factors    && NUM OF ROWS IN THE CORE CCD DESIGN
MATRIX
    T = (2 * num_factors) + center_pts
    Q = (SQRT(F + T) - SQRT(F))**2
    RETURN(((Q * F)/4)**(1/4))

FUNCTION validate
PARAMETERS valid_num,info_msg
    DO CASE
        CASE valid_num = 1
            IF ((num_rows = 0).OR.(num_rows > 1))
                RETURN(.T.)
            ENDIF
            valid_msg = "FACTORS MUST BE 0, 2 OR MORE"
        CASE valid_num = 2
            IF ((num_rows >= 2).AND.(num_rows <= 6))
                RETURN(.T.)
            ENDIF
            valid_msg = "FACTORS MUST BE BETWEEN 2 AND 6"
        CASE valid_num = 3
            IF ((sel = 2).AND.(level = 3))
                compare_val = 1
                valid_msg = "CENTER POINTS MUST BE 1 OR GREATER FOR
CCD"
            ELSE

```

```

        compare_val = 0
        valid_msg = "CENTER POINTS CANNOT BE NEGATIVE"
    ENDIF
    IF num_center_pts >= compare_val
        RETURN(.T.)
    ENDIF
CASE valid_num = 4
    IF num_reps >= 0
        RETURN(.T.)
    ENDIF
    valid_msg = "REPLICATIONS CANNOT BE NEGATIVE"
CASE valid_num = 5
    IF ((num_groups >= 1).AND.(num_groups <= num_rows))
        RETURN(.T.)
    ENDIF
    valid_msg = "# OF GROUPS MUST BE BETWEEN ONE AND # OF
FACTORS"
ENDCASE
valid_msg = "ERROR -> " + valid_msg + " <PRESS RETURN>"
@ 0,1 SAY valid_msg + SPACE(80 - len(valid_msg))
wait_key = inkey(0)
info_msg = info_msg + " <ENTER NEW VALUE>"
@ 0,1 SAY info_msg + SPACE(80 - len(info_msg))
RETURN(.F.)

```

GETRES.PRG

```

PROCEDURE p_getres
PARAMETERS level,num_rows,res,runs,num_elements
DO CASE
CASE num_rows = 2
    res[1] = 0
    runs[1] = 4
CASE num_rows = 3
    num_elements = 2
    res[1] = 3
    res[2] = 0
    runs[1] = 4
    runs[2] = 8
CASE num_rows = 4
    num_elements = 2
    res[1] = 4
    res[2] = 0
    runs[1] = 8
    runs[2] = 16
CASE num_rows = 5
    num_elements = 3
    res[1] = 3
    res[2] = 5
    res[3] = 0

```

```

    runs[1] = 8
    runs[2] = 16
    runs[3] = 32
CASE num_rows = 6
    num_elements = 4
    res[1] = 3
    res[2] = 4
    res[3] = 6
    res[4] = 0
    runs[1] = 8
    runs[2] = 16
    runs[3] = 32
    runs[4] = 64
CASE num_rows = 7
    num_elements = 3
    res[1] = 3
    res[2] = 4
    res[3] = 7
    runs[1] = 8
    runs[2] = 16
    runs[3] = 64
CASE num_rows = 8
    num_elements = 3
    res[1] = 3
    res[2] = 4
    res[3] = 5
    runs[1] = 8
    runs[2] = 16
    runs[3] = 64
CASE (num_rows >= 9).AND.(num_rows <= 11)
    num_elements = 2
    res[1] = 3
    res[2] = 4
    runs[1] = 16
    runs[2] = 32
CASE num_rows = 12
    res[1] = 3
    runs[1] = 12
CASE (num_rows >= 13).AND.(num_rows <= 16)
    res[1] = 3
    runs[1] = 16
CASE (num_rows >= 17).AND.(num_rows <= 20)
    res[1] = 3
    runs[1] = 20
CASE (num_rows >= 21).AND.(num_rows <= 24)
    res[1] = 3
    runs[1] = 24
CASE (num_rows >= 25).AND.(num_rows <= 28)
    res[1] = 3
    runs[1] = 28

```

```

CASE (num_rows >= 29).AND.(num_rows <= 32)
  res[1] = 3
  runs[1] = 32
CASE (num_rows >= 33).AND.(num_rows <= 36)
  res[1] = 3
  runs[1] = 36
CASE (num_rows >= 37).AND.(num_rows <= 40)
  res[1] = 3
  runs[1] = 40
ENDCASE
RETURN

```

FACSET.PRG

```

*
* FACSET.PRG
*
PROCEDURE p_facset
PARAMETERS level,num_rows
num_fields = 3
retrieve = .F.
data_entry = .F.
factor_selection = .F.
exit_facset = .F.
save_facset_screen = savescreen(2,9,8,26)
DO WHILE !exit_facset
  prompt_size = 16
  @ 2,9,8,26 BOX sl_box
  @ 24,0 SAY SPACE(80)
  @ 3,10 PROMPT menu_pad("Save",prompt_size);
    MESSAGE "SELECT TO SAVE THE CURRENT DATA ENTRY
SESSION"
  @ 4,10 PROMPT menu_pad("Retrieve",prompt_size);
    MESSAGE "SELECT TO RETRIEVE A PREVIOUSLY SAVED
FILE"
  @ 5,10 PROMPT menu_pad("Data Entry",prompt_size);
    MESSAGE "SELECT TO ENTER NEW DATA OR EDIT RETRIEVED
DATA"
  @ 6,10 PROMPT menu_pad("Factor Selection",prompt_size);
    MESSAGE "SELECT TO RETAIN FACTORS FOR FUTHER
ANALYSIS"
  @ 7,10 PROMPT menu_pad("Exit",prompt_size);
    MESSAGE "SELECT TO EXIT THIS MENU"
  choice = 1
  MENU TO choice
  DO CASE
    CASE choice = 1
      IF (retrieve.OR.data_entry)
        * get the file name of the save file
        overwrite = .F.

```

```

        prompt_msg = "INPUT THE FACTOR SETTINGS FILE NAME"
        filename =
input_filename(".FAC",@overwrite,prompt_msg)
        IF overwrite
            DO p_write_it  && write the data to disk using
the filename
        ENDIF
    ELSE
        @ 0,1 SAY "Error -- You must have DATA present" + ;
            " to SAVE a file <PRESS RETURN>"
        key_wait = inkey(0)
        @ 0,0 SAY SPACE(80)
    ENDIF
CASE choice = 2
    retrieve = .F.  && for use in the Factor Selection
menu item
    @ 0,1 SAY "INPUT THE FACTOR SETTINGS FILE NAME"
    filename = get_filename("*.FAC")
    @ 0,0 SAY SPACE(80)
    IF FILE(filename)
        DO p_read_it                && read the saved data from
file
        retrieve = .T.
    ELSE
        @ 0,1 SAY "Error -- This file does not exist" + ;
            " <PRESS RETURN>"
        key_wait = inkey(0)
        @ 0,0 SAY SPACE(80)
    ENDIF
CASE choice = 3
    IF p_data_entry() = 0                && operate on the data

        data_entry = .F.                && abnormal ending
    ELSE
        data_entry = .T.
    ENDIF
CASE choice = 4                && select the correct
factors
    * a file must exist to select factors
    IF retrieve
        factor_selection = .F.
        DECLARE ar[num_rows]
        FOR i = 1 TO num_rows
            ar[i] = " "
            ar[i] = " " + chr(186) + name[i] + " " +
ltrim(str(low[i])) + ;
                " " + ltrim(str(high[i])) + " " +
vlabel[i]
        NEXT
        savebrow = savescreen(11,10,18,62)

```

```

        afill(asterisk,.F.)  && fill with false values (no
selection)
        sel = abrowse(ar,num_rows,END_KEY,11,10,18,62)
        restscreen(11,10,18,62,savebrow)
        IF sel > 0  && ESCAPE KEY NOT PRESSED IN SELECTION
            factor_selection = .T.
        ENDIF
    ELSE
        @ 0,1 SAY "Error -- You must have a file retrieved"
+ ;
        " to SELECT from <PRESS RETURN>"
        key_wait = inkey(0)
        @ 0,0 SAY SPACE(80)
    ENDIF
    CASE choice = 5
        exit_facset = .T.
    ENDCASE
ENDDO
* restore the screen for the facset screen
restscreen(2,9,8,26,save_facset_screen)
RETURN

PROCEDURE p_write_it
*
* DETERMINE THE NUMBER OF FACTORS REMAINING FROM FACTOR
SELECTION
*
    num_factors = 0
    IF factor_selection
        FOR i = 1 to num_rows
            IF asterisk[i]
                num_factors = num_factors + 1
            ENDIF
        NEXT
    ELSE
        num_factors = num_rows
    ENDIF
    * DO NOT ALLOW A FILE TO BE WRITTEN IF THERE ARE NO
FACTORS SELECTED
    IF factor_selection.AND.num_factors = 0
        @ 0,1 SAY "NO FILE WRITTEN BECAUSE NO FACTORS WERE
SELECTED <RETURN>"
        wait_key = inkey(0)
        @ 0,0 SAY SPACE(80)
        factor_selection = .F.
        RETURN
    ENDIF

    handle = FCREATE(filename)
    eol = chr(13) + chr(10)

```

```

comma = ","
wbuffer = ltrim(str(num_factors)) + eol
FWRITE(handle,wbuffer)
wbuffer = ""
FOR i = 1 TO num_rows
  IF factor_selection
    write_value = asterisk[i]
  ELSE
    write_value = .T.
  ENDIF
  IF write_value      && IF TRUE THEN ALLOW THE WRITE TO
TAKE PLACE
    wbuffer = ltrim(str(low[i])) + comma + ;
               ltrim(str(high[i])) + comma + vlabel[i]
    FWRITE(handle,wbuffer)
    IF i < num_rows
      FWRITE(handle,eol)
    ENDIF
  ENDIF
  wbuffer = ""
NEXT
factor_selection = .F.
FCLOSE(handle)
RETURN

```

```

PROCEDURE p_read_it
PRIVATE parse_it[3]
LINE_SIZE = 512
line = SPACE(LINE_SIZE)
comma = ","
afill(parse_it," ")
handle = FOPEN(filename)
first_read = .T.
num_fields = 1
DO WHILE freadln(handle,@line,LINE_SIZE)
  buffer = line
  FOR i = 1 to num_fields
    comma_pos = AT(comma,buffer)
    parse_it[i] = substr(buffer,1,(comma_pos - 1))
    buffer = substr(buffer,-(len(buffer) - comma_pos))
  NEXT
  IF first_read
    num_rows = val(buffer)
    PUBLIC
name[num_rows],low[num_rows],med[num_rows],high[num_rows]
    PUBLIC vlabel[num_rows],asterisk[num_rows]
    num_fields = 2
    j = 0
    first_read = .F.
  ELSE

```



```

        j = j + 1
        name[j]   = "X" + ltrim(str(j))
        low[j]    = val(parse_it[1])
        high[j]   = val(parse_it[2])
        vlabel[j] = trim(buffer)
    ENDIF
ENDDO
FCLOSE(handle)
RETURN

FUNCTION p_data_entry
@ 0,1 SAY "SELECT OR ENTER A DESIGN FILENAME"
des_filename = get_filename("*.DES")
@ 0,0 SAY SPACE(80)
IF !FILE(des_filename)
    @ 0,1 SAY " ERROR -- DESIGN FILE DOES NOT EXIST " + ;
        "PRESS ANY KEY TO CONTINUE"
    wait_key = inkey(0)
    @ 0,0 SAY SPACE(80)
    RETURN(0)
ELSE
    DO p_flagval WITH des_filename
    group_num = flag_val[1]
    num_rows  = flag_val[2]    && get the number of factors
ENDIF
@ 0,1 SAY "RETURN TO ENTER DATA - ARROWS TO MOVE - END TO
LEAVE"
PUBLIC
name[num_rows],low[num_rows],med[num_rows],high[num_rows]
PUBLIC vlabel[num_rows],asterisk[num_rows]
afill(name,SPACE(20))
afill(low,0)
afill(med,0)
afill(high,0)
afill(vlabel,SPACE(20))
afill(asterisk,.F.) && fill with false values (no
selection)
pos = 1
FOR i = 1 TO num_rows
    name[i] = "X" + ltrim(str(i))
NEXT
current_field = 1
current_row = 1
key = 0
* save the area around the input area
savewindow = savescreen(18,1,22,63)
@ 18,1 SAY CHR(213) + REPLICATE(CHR(205),61) + CHR(184)
@ 19,1 SAY CHR(179) + " VARIABLE NAME          LOW
HIGH" + ;
        "          VARIABLE LABEL    " + CHR(179)

```

```

@ 20,1 TO 22,63  && Draw a box around input field
@ 20,1 SAY CHR(195)  && LEFT CORNER BOX EXTENDER
@ 20,63 SAY CHR(180)  && RIGHT CORNER BOX EXTENDER
*@ 10,50 SAY SPACE(10)  && Fill in the block with spaces
DO WHILE key != END_KEY
    key = 0
    SET CURSOR OFF
    * select cell to edit or escape
    DO WHILE (key != ENTER.AND.key != END_KEY)
        @ 21,2 SAY menu_pad(name[current_row],21)
        FOR field_num = 1 TO num_fields
            SET COLOR TO &norm
            @ 20,23+(field_num-1) * 10 SAY chr(194)
            @ 21,23+(field_num-1) * 10 SAY chr(179)
            @ 22,23+(field_num-1) * 10 SAY chr(193)
            DO get_string_proc
            IF field_num = current_field
                SET COLOR TO &enh
            ELSE
                SET COLOR TO &norm
            ENDIF
            @ 21,24+(field_num-1) * 10 SAY &array_name[current_row]
        ;
        PICT pic_string
    NEXT

    key = inkey(0)  && Wait until a key is pressed
    * ESC PRESSED THEN EXIT AND RETURN 0
    IF key = ESC
        @ 0,0 SAY SPACE(80)
        restscreen(18,1,22,63,savewindow)
        RETURN(0)
    ENDIF
    * Remove the highlight from the current field
    SET COLOR TO &norm
    @ 21,24+(current_field-1) * 10 SAY
&array_name[current_row] ;
    PICT pic_string

DO CASE
    CASE key = DOWN_ARROW
        DO dn_arrow_proc
    CASE key = UP_ARROW
        DO up_arrow_proc
    CASE key = LT_ARROW
        DO lt_arrow_proc
    CASE key = RT_ARROW
        DO rt_arrow_proc
    CASE ((key >= asc('0').AND.key <= asc('9')).OR. ;
        (key >= asc('A').AND.key <= asc('z')))

```

```

        EXIT
    ENDCASE
ENDDO
SET CURSOR ON
IF key <> END_KEY
    * starting edit mode
    IF key != ENTER
        KEYBOARD chr(key)
    ENDIF
    field_num = current_field
    DO get_string_proc
    get_string = &array_name[current_row]
    @ 21,24+(current_field-1)*10 GET get_string PICT
pic_string
    READ
    IF UPDATED()  && if there was a change in the string
then update
        &array_name[current_row] = get_string
    ENDIF
ENDIF
ENDDO
SET CURSOR ON
@ 0,0 SAY SPACE(80)
restscreen(18,1,22,63,savewindow)
RETURN(1)  && IF NORMAL END RETURN 1

```

```

PROCEDURE get_string_proc
    PUBLIC array_name,pic_string
    pic_string = "@S10@B 9999999999999999.9999"
    DO CASE
        CASE field_num = 1
            array_name = "low"
        CASE field_num = 2
            array_name = "high"
        CASE field_num = 3
            array_name = "vlabel"
            pic_string = "XXXXXXXXXXXXXXXXXXXX"
    ENDCASE
RETURN

```

```

PROCEDURE dn_arrow_proc
    * If not on last line go down by fields_line
    IF current_row < num_rows
        current_row = current_row + 1
    ENDIF
RETURN

```

```

PROCEDURE up_arrow_proc
    * If not on top line

```

```

    IF current_row > 1
        current_row = current_row - 1
    ENDIF
RETURN

```

```

PROCEDURE lt_arrow_proc
    * If not at first field get...
    IF current_field > 1
        current_field = current_field - 1
    ELSEIF current_row > 1    && current_field = 1
        * Go to last column on previous row
        current_field = num_fields
        current_row = current_row - 1
    ENDIF
RETURN

```

```

PROCEDURE rt_arrow_proc
    * If not at last field get...
    IF current_field < num_fields
        current_field = current_field + 1
    ELSEIF current_row < num_rows    && current_field = 1
        * Go to first column on next row
        current_field = 1
        current_row = current_row + 1
    ENDIF
RETURN

```

RAWDAT.PRG

```

PROCEDURE p_rawdat
    @ 0,1 SAY "SELECT OR ENTER A DESIGN FILENAME"
        des_filename = get_filename("*.DES")
    @ 0,0 SAY SPACE(80)
        IF !FILE(des_filename)
            @ 0,1 SAY " ERROR -- DESIGN FILE DOES NOT EXIST
<RETURN>"
                wait_key = inkey(0)
            @ 0,0 SAY SPACE(80)
            RETURN
        ENDIF

    * make sure that exp.des is current filename
    IF UPPER(des_filename) != "EXP.DES"
        COPY FILE &des_filename TO exp.des
    ENDIF

    * READ .DES FOR # OF GROUPS AND # OF FACTORS
    DO p_flagval WITH des_filename
    groups      = flag_val[1]
    num_factors = flag_val[2]

```

```

IF FILE("exp.fac")
  ERASE
ENDIF
@ 0,1 SAY "SELECT OR ENTER A FACTORS FILENAME"
fac_filename = get_filename("*.FAC")
@ 0,0 SAY SPACE(80)
IF !FILE(fac_filename)
  @ 0,1 SAY " ERROR -- FACTORS FILE DOES NOT EXIST
<RETURN>"
  wait_key = inkey(0)
  @ 0,0 SAY SPACE(80)
  RETURN
ENDIF

* check to see that .des and .fac have the same
number of factors
handle = FOPEN(fac_filename)
fnum_factors = val(freadline(handle))
FCLOSE(handle)
IF fnum_factors != num_factors
  @ 0,1 SAY " ERROR -- NOT THE SAME NUMBER OF FACTORS
IN .DES " + ;
  " AND .FAC SELECTED <RETURN>"
  wait_key = inkey(0)
  @ 0,0 SAY SPACE(80)
  RETURN
ENDIF

COPY FILE &fac_filename TO exp.fac
IF groups > 0 && GROUP SCREENING
  DECLARE group_num[num_factors] && create group
number array
  afill(group_num,0)
  @ 24,0 SAY SPACE(80)
  FOR i = 1 TO num_factors
    @ 24,0 SAY "Enter the group number for factor " +
;
    "X" + ltrim(str(i)) + " ==> " ;
    GET group_num[i] PICT "999" ;
    VALID ((group_num[i] > 0).AND.(group_num[i] <=
groups))
  READ
  IF LASTKEY() = ESC
    @ 0,0 SAY SPACE(80)
    @ 24,0 SAY SPACE(80)
    RETURN
  ENDIF
NEXT
* write out the group number file
handle = FCREATE("exp.grp")

```

```

        FOR i = 1 TO num_factors
            buffer = LTRIM(STR(group_num[i]),) + " "
            FWRITE(handle,@buffer)
        NEXT
        FCLOSE(handle)
        @ 24,0 SAY SPACE(80)
    ENDIF  && for group screening

    IF FILE("clipraw.exe")
        * C CODE TO GROUP RAW DATA FILE
        SAVE SCREEN
        CLEAR
        RUN clipraw
        RESTORE SCREEN
        overwrite = .F.
        prompt_msg = "PLEASE ENTER THE RAW DATA FILE NAME"
        raw_filename =
input_filename(".RAW",@overwrite,prompt_msg)
        IF !EMPTY(raw_filename) .AND. overwrite
            COPY FILE exp.raw TO &raw_filename
        ENDIF
    ELSE
        @ 0,1 SAY " ERROR -- FILE > clipraw.exe < DOES NOT
EXIST <RETURN>"
        wait_key = inkey(0)
        @ 0,0 SAY SPACE(80)
        RETURN
    ENDIF
RETURN

```

REGPROC.PRG

```

*
* REGPROC.PRG
*
PROCEDURE p_varsel
PARAMETERS des_filename
    SAVE SCREEN
    CLEAR
    cvarsel(des_filename)  && reads des_filename makes
selection and
    RESTORE SCREEN          && writes exp.des as current file
    * get the file name of the save file
    overwrite = .F.
    prompt_msg = "INPUT THE DESIGN FILE NAME TO SAVE SELECTION
RESULTS"
    filename = input_filename(".DES",@overwrite,prompt_msg)
    IF !EMPTY(filename) .AND. overwrite
        COPY FILE exp.des TO &filename
    ENDIF

```

RETURN

PROCEDURE p_response

PUBLIC res_filename

IF FILE("exp.res")

ERASE exp.res

ENDIF

* THIS ENSURES THAT ONLY EXP.RES IS THE CURRENT FILE FOR
REGRESSION

IF FILE("trans.res")

ERASE trans.res

ENDIF

respond = .T.

@ 24,0 SAY SPACE(80)

@ 24,1 SAY "DO YOU WANT TO RETRIEVE *.RES FILE FROM DISK <Y>
?" ;

GET respond PICT "Y"

READ

@ 24,0 SAY SPACE(80)

IF respond

* read .RES file and create y_vector

@ 0,0 SAY SPACE(80)

@ 0,1 SAY "INPUT THE RESPONSE FILE NAME"

res_filename = get_filename("*.RES")

@ 0,0 SAY SPACE(80)

IF !FILE(res_filename)

@ 0,1 SAY "ERROR -> RESPONSE FILE DOES NOT EXIST

<RETURN>"

wait_key = inkey(0)

@ 0,0 SAY SPACE(80)

RETURN

ELSE

* MAKE THE RETRIEVE FILE THE CURRENT RESPONSE FILE FOR
REGRESSION

COPY FILE &res_filename TO exp.res

ENDIF

ELSE

@ 0,1 SAY "INPUT THE DESIGN FILE NAME THAT CORRESPONDS TO
THE RESPONSES"

des_filename = get_filename("*.DES")

@ 0,0 SAY SPACE(80)

IF !FILE(des_filename)

@ 0,1 SAY "ERROR -> DESIGN FILE DOES NOT EXIST <RETURN>"

wait_key = inkey(0)

@ 0,0 SAY SPACE(80)

RETURN

ELSE

DO p_flagval WITH des_filename

num_runs = flag_val[3]

PUBLIC y_vector[num_runs]

```

    afill(y_vector,0.000000)
    @ 0,0 SAY SPACE(80)
    @ 0,1 SAY "INPUT THE RESPONSES IN RUN ORDER"
    FOR i = 1 TO num_runs
        @ 24,1 SAY "FOR RUN # " + ltrim(str(i)) + ;
            " ENTER RESPONSE VALUE => " ;
            GET y_vector[i]
        READ
        IF LASTKEY() = ESC
            @ 0,0 SAY SPACE(80)
            @ 24,0 SAY SPACE(80)
            RETURN
        ENDIF
    NEXT
    handle = FCREATE("EXP.RES")
    eol = chr(13) + chr(10)
    wbuffer = ltrim(str(num_runs)) + eol
    FWRITE(handle,wbuffer)
    wbuffer = ""
    FOR i = 1 TO num_runs
        wbuffer = ltrim(str(y_vector[i]))
        FWRITE(handle,wbuffer)
        IF i < num_runs
            FWRITE(handle,eol)
        ENDIF
        wbuffer = ""
    NEXT
    FCLOSE(handle)
    overwrite = .F.
    prompt_msg = "INPUT THE RESPONE FILE NAME TO SAVE
INPUTS"
    res_filename =
input_filename(".RES",@overwrite,prompt_msg)
    * IF NO SAVED FILENAME GIVEN THEN DO NOT COPY TO SAVE
.RES FILE
    IF !EMPTY(res_filename) .AND. overwrite
        COPY FILE exp.res TO &res_filename
    ENDIF
ENDIF
ENDIF
RETURN

PROCEDURE p_transform
@ 0,1 SAY "TRANSFORMATIONS RESPONSE DATA USING THE FOLLOWING
FUNCTIONS"
exit_trans = .F.
save_trans_screen = savescreen(2,9,14,28)
prompt_size = 18
DO WHILE !exit_trans
    @ 2,9,14,28 BOX sl_box

```



```

@ 24,0 SAY SPACE(80)
@ 3,10 PROMPT menu_pad("Y**ALPHA",prompt_size);
      MESSAGE "SELECT FOR POWER TRANSFORMATION OF Y"
@ 4,10 PROMPT menu_pad("LN(Y)",prompt_size);
      MESSAGE "SELECT FOR NATURAL LOG OF Y"
@ 5,10 PROMPT menu_pad("LOG10(Y)",prompt_size);
      MESSAGE "SELECT FOR LOG BASE 10 OF Y"
@ 6,10 PROMPT menu_pad("ARCSIN SQRT(Y)",prompt_size);
      MESSAGE "SELECT FOR ARCSIN SQUARE ROOT OF Y"
@ 7,10 PROMPT menu_pad("LN( (1+Y)/(1-Y) )",prompt_size);
      MESSAGE "SELECT FOR NATURAL LOG OF (1+Y)/(1-Y)"
@ 8,10 PROMPT menu_pad("(1/Y)",prompt_size);
      MESSAGE "SELECT FOR INVERSE OF Y"
@ 9,10 PROMPT menu_pad("SQRT(Y)",prompt_size);
      MESSAGE "SELECT FOR THE SQUARE ROOT OF Y"
@ 10,10 PROMPT menu_pad("Y**2",prompt_size);
      MESSAGE "SELECT FOR Y SQUARED"
@ 11,10 PROMPT menu_pad("LN(B - Y)",prompt_size);
      MESSAGE "SELECT FOR NATURAL LOG OF (B - Y)"
@ 12,10 PROMPT menu_pad("Original Values",prompt_size);
      MESSAGE "SELECT FOR THE ORIGINAL RESPONSE VALUES"
@ 13,10 PROMPT menu_pad("Exit",prompt_size);
      MESSAGE "SELECT TO EXIT THIS MENU"

choice = 1
MENU TO choice
t_value = 0.00000
DO CASE
  CASE choice = 1
    @ 24,1 SAY "ENTER ALPHA VALUE FOR POWER TRANSFORMATION
=>" ;
      GET t_value
      READ
      @ 24,0 SAY SPACE(80)
      CASE choice = 9
        @ 24,1 SAY "ENTER B VALUE FOR NATURAL LOG
TRANSFORMATION =>" ;
          GET t_value
          READ
          @ 24,0 SAY SPACE(80)
      ENDCASE
      IF LASTKEY() != ESC
        IF choice = 11
          exit_trans = .T.
        ELSE
          IF choice = 10
            * make the current file exp.res instead of trans.res
            IF FILE("trans.res")
              ERASE trans.res
            ENDIF
            res_filename = "exp.res"

```

```

ELSE
    res_filename = "trans.res"
    PRIVATE carray[3]
    carray[1] = "exp.res"
    carray[2] = choice
    carray[3] = t_value
    SAVE SCREEN
    CLEAR
    * call to C function to do transformations
    ctrans(carray)
    RESTORE SCREEN
ENDIF
ENDIF
ENDIF
ENDDO
@ 0,0 SAY SPACE(80)
* restore the screen for the facet screen
restscreen(2,9,14,28,save_trans_screen)
RETURN

```

BROW.PRG

```

FUNCTION abrowse
PARAMETERS
ar,num_elems,key_var,start_row,start_col,end_row,end_col
PRIVATE
num_disp_rows,floor,ceiling,key,highlight,width,cur_disp_row
s
IF num_elems = 0
    RETURN (0)
ENDIF
SET CURSOR OFF
@ start_row,start_col,end_row,end_col BOX SL_BOX
num_disp_rows = end_row - start_row - 1
width = end_col - start_col - 1
floor = 1
highlight =1
cur_disp_rows = afill_box(ar,start_row,start_col,end_row,;
                        end_col,floor)
ceiling = cur_disp_rows + floor - 1
IF ceiling < num_elems
    SET COLOR TO /W
    @ end_row,start_col SAY chr(25)
    SET COLOR TO W/
ENDIF
SET COLOR TO /W
* Highlight active element
@ start_row + highlight,start_col + 1 SAY;
    pad(ar[floor + highlight - 1],width)
SET COLOR TO W/

```

```

key = inkey(0)

* key_var is either ENTER or ESC to activate the ENTER case
DO WHILE (key != ESC).AND.(key != key_var)
  * Remove highlight from active element
  @ start_row + highlight, start_col + 1 SAY;
  pad(ar[floor + highlight - 1], width)
DO CASE
  CASE key = ENTER
    IF asterisk[floor + highlight - 1] && if element is
active
      * deactivate the element
      asterisk[floor + highlight - 1] = .F.
      ar[floor + highlight - 1] = ;
      STUFF(ar[floor + highlight - 1], 1, 2, " ")
    ELSE
      * activate the element
      asterisk[floor + highlight - 1] = .T.
      ar[floor + highlight - 1] = ;
      STUFF(ar[floor + highlight - 1], 1, 2, ">>")
    ENDIF
  CASE key = UP_ARROW
    IF highlight > 1
      highlight = highlight - 1
    ELSE
      IF floor > 1
        floor = floor - 1
        scroll(start_row + 1, start_col + 1, ;
              end_row - 1, end_col - 1, -1)
        * This leaves highlight row empty
        cur_disp_rows = min(num_disp_rows, ;
                           num_elems - floor + 1)
        ceiling = floor + cur_disp_rows - 1
      ENDIF
    ENDIF
  CASE key = DOWN_ARROW
    IF highlight < cur_disp_rows
      highlight = highlight + 1
    ELSE
      IF ceiling < num_elems
        floor = floor + 1
        scroll(start_row + 1, start_col + 1, ;
              end_row - 1, end_col - 1, 1)
        * This leaves highlight row empty
        cur_disp_rows = min(num_disp_rows, num_elems -
floor + 1)
        ceiling = floor + cur_disp_rows - 1
      ENDIF
    ENDIF
  CASE key = PG_UP

```

```

    IF floor > 1
        floor = max(floor - num_disp_rows,1)
        ceiling = afill_box(ar,start_row,start_col,end_row,;
                           end_col,floor)
        cur_disp_rows = ceiling - floor + 1
    ENDIF

CASE key = PG_DOWN
    IF ceiling < num_elems
        floor = min(floor + num_disp_rows,num_elems)
        cur_disp_rows = afill_box(ar,start_row,start_col,;
                                   end_row,end_col,floor)
        ceiling = cur_disp_rows + floor - 1
        highlight = min(highlight,cur_disp_rows)
    ENDIF

CASE key = HOME
    highlight = 1
CASE key = END_KEY
    highlight = cur_disp_rows
CASE key = CTRL_HOME
    highlight = 1
    floor = 1
    cur_disp_rows = afill_box(ar,start_row,start_col,;
                               end_row,end_col,floor)
    ceiling = cur_disp_rows + floor - 1
CASE key = CTRL_END
    IF ceiling = num_elems
        highlight = cur_disp_rows
    ELSE
        floor = num_elems - num_disp_rows + 1
        cur_disp_rows = afill_box(ar,start_row,start_col,;
                                   end_row,end_col,floor)
        ceiling = cur_disp_rows + floor - 1
        highlight = cur_disp_rows
    ENDIF
ENDCASE
* Highlight active element
SET COLOR TO /W
@ start_row + highlight,start_col + 1 ;
  SAY pad(ar[floor + highlight - 1],width)
SET COLOR TO W/

IF floor > 1
    SET COLOR TO /W
    @ start_row,start_col SAY chr(24)
    SET COLOR TO W/
ELSE
    @ start_row,start_col SAY chr(218)
ENDIF

```

```

IF ceiling < num_elems
  SET COLOR TO /W
  @ start_row,start_col SAY chr(25)
  SET COLOR TO W/
ELSE
  @ start_row,start_col SAY chr(192)
ENDIF

key = inkey(0)
ENDDO
SET CURSOR OFF
RETURN(IF(lastkey() = ESC,0,floor + highlight -1))

FUNCTION afill_box
PARAMETERS ar,start_row,start_col,end_row,end_col,floor
PRIVATE num_disp,num_rows,i,width

num_rows = end_row - start_row - 1
width = end_col - start_col - 1
num_disp = min(num_rows,num_elems - floor + 1)

FOR i = 1 TO num_disp
  @ start_row + i,start_col + 1 SAY pad(ar[floor + i -
1],width)
NEXT
FOR i = num_disp + 1 TO num_rows
  @ start_row + i,start_col + 1 SAY space(width)
NEXT

RETURN(num_disp)

FUNCTION pad
PARAMETERS str,width
IF len(str) > width
  str = substr(str,1,width)
ELSE
  str = str + space(width - len(str))
ENDIF

RETURN (str + space(width - len(str)))

CONSTS.PRG

FUNCTION init_consts

PUBLIC
SL_BOX,BOX1,BOX2,ESC,UP_ARROW,PG_UP,DOWN_ARROW,PG_DOWN,CTRL_
END

```

```

PUBLIC
END_KEY, HOME, CTRL_HOME, ENTER, F1, F2, F3, F4, F5, F6, F7, F8, F9, F10
PUBLIC LT_ARROW, RT_ARROW, CTRL_W, CTRL_Y, CTRL_T
PUBLIC AC_IDLE, AC_TOP, AC_BOT, AC_EXCEP, AC_NOITEM, AC_ABORT
PUBLIC AC_CONTINUE, BRIGHT, NORM, ENH, AC_GO_MATCH, AC_SELECT

* achoices modes
AC_IDLE    = 0
AC_TOP     = 1
AC_BOT     = 2
AC_EXCEP   = 3
AC_NOITEM  = 4

* achoice return values
AC_ABORT   = 0
AC_SELECT  = 1
AC_CONTINUE = 2
AC_GO_MATCH = 3

* color values for BW screen
BRIGHT = "W+/"
NORM    = "W/"
ENH     = "/W"

* single line box
SL_BOX = chr(218) + chr(196) + chr(191) + chr(179) +
chr(217) + ;
          chr(196) + chr(192) + chr(179)
BOX1    = chr(213) + chr(205) + chr(184) + chr(179) +
chr(217) + ;
          chr(196) + chr(192) + chr(179)
BOX2    = chr(201) + chr(205) + chr(187) + chr(186) +
chr(188) + ;
          chr(205) + chr(200) + chr(186)

* some key values from INKEY
ESC      = 27
UP_ARROW = 5
PG_UP    = 18
DOWN_ARROW = 24
PG_DOWN  = 3
CTRL_END = 23
END_KEY  = 6
HOME     = 1
CTRL_HOME = 29
ENTER    = 13
LT_ARROW = 19
RT_ARROW = 4
F1       = 28
F2       = -1

```

```

F3          = -2
F4          = -3
F5          = -4
F6          = -5
F7          = -6
F8          = -7
F9          = -8
F10         = -9
CTRL_W      = 23
CTRL_Y      = 25
CTRL_T      = 20

```

```
RETURN(.T.)
```

FUNC.PRG

```

FUNCTION p_flagval
PARAM des_filename
PUBLIC flag_val[12]
handle = FOPEN(des_filename)
current_string = freadline(handle)
FOR i = 1 to 11
    temp_string      = LTRIM(current_string)
    blank_pos        = AT(" ",temp_string)
    flag_val[i]      = VAL(LEFT(temp_string,blank_pos-1))
    current_string = SUBSTR(temp_string,blank_pos)
NEXT
FCLOSE(handle)
flag_val[12]      = VAL(current_string)
RETURN(.T.)

FUNCTION input_filename
PARAMETERS extension_string,overwrite,prompt_msg
PRIVATE extension_string,filename,correct
    filename = SPACE(12)
    respond = .T.
    @ 24,0 SAY SPACE(80)
    @ 24,1 SAY "DO YOU TO SAVE " + extension_string + ;
              " FILE TO DISK <Y> ?" GET respond PICT "Y"

    READ
    @ 24,0 SAY SPACE(80)
    IF respond
        @ 0,1 SAY prompt_msg
        correct = .F.
        DO WHILE .NOT.correct && loop to get the filename to
save
            filename = SPACE(8)
            @ 24,1 SAY "Input the " + extension_string + ;
                    " file name (up to 8 chars) => " ;
            GET filename PICT "@! ANNNNNNN"

```

```

        READ
        IF LASTKEY() = ESC
        @ 0,0 SAY SPACE(80)
        RETURN(SPACE(12))
        ENDIF
        filename = ALLTRIM(filename) + extension_string
        @ 24,0 SAY SPACE(80)
        say_string = "Is the filename " + filename + " Correct
- <Y>"
        @ 24,1 SAY say_string GET correct PICT "Y"
        READ
        IF LASTKEY() = ESC
        @ 0,0 SAY SPACE(80)
        RETURN(SPACE(12))
        ENDIF
        @ 24,0 SAY SPACE(80)
        IF correct
        * CHECK TO SEE IF FILE EXISTS ON DISK
        IF FILE(filename)
        @ 24,1 SAY "FILE =>" + filename + "<= EXISTS -- DO
YOU WANT " +;
                                "TO OVERWRITE <N> ?" GET overwrite PICT
"Y"
                READ
                IF LASTKEY() = ESC
                @ 0,0 SAY SPACE(80)
                RETURN(SPACE(12))
                ENDIF
                @ 24,0 SAY SPACE(80)
                IF !overwrite
                correct = .F.      && ASK FOR ANOTHER FILE NAME
                ENDIF
            ELSE
                overwrite = .T.      && SEND BACK POSITIVE WRITE
MESSAGE
                ENDIF
            ENDIF
        ENDDO
        @ 0,0 SAY SPACE(80)
        ENDIF
        RETURN(filename)

```

```

FUNCTION get_filename
PARAMETERS dir_search_string
PRIVATE dir_search_string,file_box[5],num_files,filename

```

```

        file_box[1] = "enter_title(sysparam)"
        file_box[2] = "rl_getfil(sysparam)"
        file_box[3] = "ok_button(sysparam)"

```



```

        file_box[4] = "cancel_button(sysparam)"
        file_box[5] = "filelist(sysparam)"

        filename = SPACE(12)
        num_files = adir(dir_search_string)
        state = 0
        IF num_files > 0
            PRIVATE files[num_files]
            adir(dir_search_string,files)
            okee_dokee = "do_it()"
            state = multibox(7,17,7,5,file_box)
        ENDIF

RETURN(IF(state = 0,SPACE(12),trim(substr(filename,1,12)) ))

FUNCTION freadln
PARAMETERS handle,buffer,max_line
PRIVATE line,BUF_SIZE,eol,num_read,SEEK_BOF,save_pos

* seek mode,absolute position
SEEK_BOF = 0
line = SPACE(max_line)
buffer = ""

* save current file position for later seek
save_pos = ftell(handle)
num_read = FREAD(handle,@line,max_line)
eol = AT(chr(13),substr(line,1,num_read))
IF eol = 0
    buffer = line
ELSE
    buffer = substr(line,1,eol-1)  && copy up to eol
    * now position file to next line (skip if) ...
    FSEEK(handle,save_pos + eol + 1,SEEK_BOF)
ENDIF
RETURN num_read != 0

FUNCTION freadline
PARAMETERS handle
ch = " "
buffer = ""
line_size = 0
num_read = fread(handle,@ch,1)
DO WHILE num_read = 1.AND.ch != chr(13)
    buffer = buffer + ch
    line_size = line_size + 1
    num_read = fread(handle,@ch,1)
ENDDO
RETURN buffer

```



```

FOR cursor = 1 TO LEN(boxarray)
    funcn = boxarray[cursor]
    x = &funcn

    * each function leaves the corner at its top left
corner
    box_row[cursor] = ROW()
    box_col[cursor] = COL()
NEXT

cursor = beg_curs
state = 2

DO WHILE state <> 0 .AND. state <> 4
    * till completed or aborted
    funcn = boxarray[cursor]

    DO CASE

        CASE state = 2
            * pointing state
            sysparam = 2
            x = &funcn

            k = INKEY(0)

            DO CASE

                CASE k = 13 .OR. jisdata(k)
                    * change to selection state
                    state = 3

                CASE k = 27
                    * abort
                    state = 0

                OTHERWISE
                    * current item becomes uncurrent
                    sysparam = 3
                    x = &funcn

                    * get a new cursor
                    cursor = matrix(cursor, k)

            ENDCASE

        CASE state = 3
            * be selected and return a new state
            sysparam = 4

```

```

        state = &funcn

        ENDCASE
    ENDDO

    RESTORE SCREEN
    RETURN state

***
*   title
***
FUNCTION enter_title
PARAMETERS sysparam

IF sysparam = 1
    @ wt + 1, wl + 2 SAY "Enter a filename "

    * set cursor for initialization
    @ wt + 1, wl + 2 SAY ""

ENDIF

RETURN 2

FUNCTION save_title
PARAMETERS sysparam

IF sysparam = 1
    * watch out for the length of file, it may exceed the
    box width (path)
    @ wt + 3, wl + 4 SAY "Save Changes to File " +
    TRIM(filename) + "?"

    * set cursor for initialization
    @ wt + 3, wl + 4 SAY ""

ENDIF

RETURN 2

***
*   get filename
***
FUNCTION rl_getfil
PARAMETERS sysparam

DO CASE

    CASE sysparam = 1 .OR. sysparam = 3

```

```

20)      @ wt + 3, wl + 2 SAY "File " + SUBSTR(filename, 1,

IF sysparam = 1
* set cursor for initialization
@ wt + 3, wl + 9 SAY ""
ENDIF

CASE sysparam = 2
* be current...hilite
SET COLOR TO I
@ wt + 3, wl + 7 SAY SUBSTR(filename, 1, 20)
SET COLOR TO

CASE sysparam = 4
* be selected...perform a GET on entry field

Note: any other 'isdata' key will also execute
selection

IF k <> 13
KEYBOARD CHR(k)
ENDIF

filename = jenter_rc(filename, wt + 3, wl + 7, 64,
"@K!S20")
SET CURSOR ON
READ
SET CURSOR OFF

IF LASTKEY() = 13 .AND. .NOT. EMPTY(filename)
* filename has been selected...go to the ok
button

filename = JPAD(filename,20)
@ wt + 3, wl + 7 SAY filename
to_ok()
ENDIF
ENDCASE

RETURN 2

***
*   file list
***
FUNCTION filelist
PARAMETERS sysparam
PRIVATE c

DO CASE

```

```

CASE sysparam = 1
    * clear the window
    scroll(wt + 1, wl + 31, wt + wh, wl + 44, 0)
    @ wt, wl + 30, wt + wh + 1, wl + 45 BOX lframe
    IF .NOT. EMPTY(files[1])
        * display the files list
        KEYBOARD CHR(27)

    achoice(wt+1, wl+32, wt+wh, wl+43, files, "ch_func", 0, asel, arel)

    ENDIF

    * set cursor for initialization
    @ wt + 1, wl + 32 SAY ""

CASE sysparam = 2

    IF EMPTY(files[1])
        * cannot cursor onto an empty list
        KEYBOARD CHR(219)
    ELSE
        * set initial relative row and array element
        asel = asel - arel + ROW() - wt - 1
        arel = ROW() - wt - 1

        * do the list selection

        C =
    achoice(wt+1, wl+32, wt+wh, wl+43, files, "ch_func", 0, asel, arel)

    IF LASTKEY() = 13
        * filename selected from list...set memvar
        filename = SUBSTR(files[c] +
SPACE(64), 1, 64)

        * display filename in entry blank
        rl_getfil(3)

        * go directly to ok button
        to_ok()
    ELSE
        IF LASTKEY() = 19
            * the system responds to CHR(19) as ^S
            KEYBOARD CHR(219)
        ELSE
            * send character to multibox
            KEYBOARD CHR(LASTKEY())
        ENDIF
    ENDIF
ENDIF

```

```

                                ENDIF
ENDCASE

RETURN 2

***
*      ok button
***
FUNCTION ok_button

PARAMETERS sysparam
PRIVATE ok, reply

ok = " Ok "
reply = 2

DO CASE

    CASE sysparam = 1 .OR. sysparam = 3
        @ wt + wh, wl + 9 SAY ok

        IF sysparam = 1
            * set cursor for initialization
            @ wt + wh, wl + 9 SAY ""
        ENDIF

    CASE sysparam = 2
        * be current...hilite
        SET COLOR TO 1
        @ wt + wh, wl + 9 SAY ok
        SET COLOR TO

    CASE sysparam = 4

        IF &okee_dokee
            * a job well done...complete the process
            reply = 4
        ENDIF

ENDCASE

RETURN reply

***
*      cancel button
***
FUNCTION cancel_button

PARAMETERS sysparam
PRIVATE can, reply

```

```

can = " Cancel "
reply = 2

DO CASE
    CASE sysparam = 1 .OR. sysparam = 3
        @ wt + wh, wl + 17 SAY can

        IF sysparam = 1
            * set cursor for initialization
            @ wt + wh, wl + 17 SAY ""

        ENDIF

    CASE sysparam = 2
        * be current...hilite
        SET COLOR TO 1
        @ wt + wh, wl + 17 SAY can
        SET COLOR TO

    CASE sysparam = 4
        * cancel selected...abort the process
        reply = 0

ENDCASE

RETURN reply

***
*   cancel button for save file box
***
FUNCTION can_button

PARAMETERS sysparam
PRIVATE can, reply

can = " Cancel "
reply = 2

DO CASE
    CASE sysparam = 1 .OR. sysparam = 3
        @ wt + wh, wl + 25 SAY can

        IF sysparam = 1
            * set cursor for initialization
            @ wt + wh, wl + 25 SAY ""

        ENDIF

    CASE sysparam = 2
        * be current...hilite

```



```

        SET COLOR TO I
        @ wt + wh, wl + 25 SAY can
        SET COLOR TO

    CASE sysparam = 4
        * cancel selected...abort the process
        reply = 0

ENDCASE

RETURN reply

***
*   no button
***
FUNCTION no_button

PARAMETERS sysparam
PRIVATE no, reply

no = " No "
reply = 2

DO CASE
    CASE sysparam = 1 .OR. sysparam = 3
        @ wt + wh, wl + 13 SAY no

        IF sysparam = 1
            * set cursor for initialization
            @ wt + wh, wl + 13 SAY ""

        ENDIF

    CASE sysparam = 2
        * be current...hilite
        SET COLOR TO I
        @ wt + wh, wl + 13 SAY no
        SET COLOR TO

    CASE sysparam = 4
        * 'No' selected...abort the process
        reply = 0
        no_save_flag = .T.

ENDCASE

RETURN reply

***
*   achoice user function

```

```

***
FUNCTION ch_func

PARAMETERS amod, sel, fel
PRIVATE k, r, srow, scol

srow = ROW()
scol = COL()

asel = sel
arel = rel
r = 2

IF asel > arel + 1
    * more files off screen up
    @ wt + 1, wl + 44 SAY CHR(24)

ELSE
    @ wt + 1, wl + 44 SAY " "
ENDIF

IF LEN(files) - asel > wh - 1 - arel
    * more files off screen down
    @ wt + wh, wl + 44 SAY CHR(25)
ELSE
    @ wt + wh, wl + 44 SAY " "
ENDIF

IF amod = 3
    k = LASTKEY()

    DO CASE

        CASE k = 27
            * escape key
            r = 0

        CASE k = 13 .OR. k = 19 .OR. k = 219
            * return or left arrow
            r = 1

        CASE k = 1
            * home key...top of list
            KEYBOARD CHR(31)

        CASE k = 6
            * end key...end of list
            KEYBOARD CHR(30)

    ENDCASE

```

ENDIF

@ M->srow, M->scol SAY ""
RETURN r

* do_it()

*

* called from the "Ok" button as "&okee_dokee"
* this function normally completes the process

FUNCTION do_it

PRIVATE done, error_str

DO CASE

* error if empty filename
CASE EMPTY(filename) && error, empty filename
KEYBOARD CHR(5)
done = .F.

OTHERWISE
done = .T.

ENDCASE

RETURN done

* relocate cursor

FUNCTION matrix

PARAMETERS old_curs, k
PRIVATE old_row, old_col, test_curs, new_curs

old_row = ROW()
old_col = box_col[old_curs]
new_curs = old_curs
test_curs = old_curs

DO CASE

CASE k = 19 .OR. k = 219
* left arrow

DO WHILE test_curs > 2
test_curs = test_curs - 1

```

        IF box_col[test_curs] < old_col .AND.
box_row[test_curs] >= old_row

        IF box_row[test_curs] < box_row[new_curs] .OR.
new_curs = old_curs
        * best so far
        new_curs = test_curs

        ENDIF
    ENDIF
ENDDO
CASE k = 4
    * right arrow

    DO WHILE test_curs < LEN(box_col)
        test_curs = test_curs + 1

        IF box_col[test_curs] > old_col .AND.
box_row[test_curs] <= old_row

        IF box_row[test_curs] > box_row[new_curs]
.OR. new_curs = old_curs
        * best so far
        new_curs = test_curs
        ENDIF
    ENDIF
ENDDO
CASE k = 5
    * up arrow

    DO WHILE test_curs > 2
        test_curs = test_curs - 1
        IF box_row[test_curs] < old_row .AND.
box_col[test_curs] <= old_col

        IF box_col[test_curs] > box_col[new_curs]
.OR. new_curs = old_curs
        * best so far
        new_curs = test_curs
        ENDIF
    ENDIF
ENDDO
CASE k = 24
    * down arrow

    DO WHILE test_curs < LEN(box_row)
        test_curs = test_curs + 1

        IF box_row[test_curs] > old_row .AND.
box_col[test_curs] >= old_col

```

```

                IF box_col[test_curs] < box_col[new_curs]
.OR. new_curs = old_curs
                * best so far
                new_curs = test_curs
            ENDIF
        ENDIF
    ENDDO
ENDCASE

RETURN new_curs

***
*   go directly to ok button
***
FUNCTION to_ok

cursor = ascan(boxarray, "ok_button(sysparam)")
KEYBOARD CHR(219)
RETURN 0

*****
*   jisdata()
*
*   determine if a key is data suitable for entry in place
*****
FUNCTION jisdata
PARAMETERS k

RETURN (M->k >= 32 .AND. M->k < 249 .AND. M->k <> 219 .AND.
CHR(M->k) <> ";")

*****
*   jenter_rc(r,c,max_len,pfunc)
*
*   entry in place
*****
FUNCTION jenter_rc

PARAMETERS org_str,r,c,max_len,pfunc
PRIVATE wk_str, keystroke

wk_str = JPAD(org_str, max_len)
SET CURSOR ON

IF .NOT. EMPTY(pfunc)
    @ r,c GET wk_str PICTURE pfunc
ELSE
    @ r,c GET wk_str
ENDIF

```

```

READ
SET CURSOR OFF

keystroke = LASTKEY()

IF keystroke = 27
    wk_str = ""
ENDIF

RETURN (TRIM(wk_str))

*****
*   jpad()
*
*   syntax: jpad( <expC>, <expN> )
*
*   return: <expC> padded with spaces so that 'len( <expC> =
*           <expN>)'
*****
FUNCTION jpad

PARAMETERS s, n

RETURN(SUBSTR(s + SPACE(n), 1, n))

```

CREATDES.C

```

/* PROGRAM CREATDES.C -- A FUNCTION FOR CLIPPER TO
DETERMINE
    WHICH DESIGN PROGRAM TO EXECUTE BASED ON LEVEL AND PASS
DATA */

# include "nandef.h"
# include "extend.h"

float alpha,desmat[65][130];
int gpscreen,factor,level;
int res,choice,orthog;
int ceps,reprs;

CLIPPER cdesign()
{
    gpscreen = _parni(1,1);
    factor    = _parni(1,2);
    level     = _parni(1,3);
    if (level == 2) {
        res    = _parni(1,4);
        ceps   = _parni(1,5);
        reprs  = _parni(1,6);
    }
}

```

```

        cdesign2(); }          /* call the create level-2
design function */
    else {
        choice = _parni(1,4);
        orthog = _parni(1,5);
        alpha = _parnd(1,6);
        ceps = _parni(1,7);
        reps = _parni(1,8);
        cdesign3();          /* call the create level-3
design function */
    }
} /* end of cdesign function */

```

LEVEL2.C

```

/* THIS PROGRAM GENERATES TWO LEVEL DESIGN MATRICES */

# include <stdio.h>
# include <math.h>

extern float alpha,desmat[130][65];
extern int gpscreen,factor,level;
extern int res,choice,orthog;
extern int ceps,reps;

/* MAIN DETERMINES # FACTORS, RESOLUTION, AND DESIGN */
cdesign2()
{
    FILE *outfile;
    int row,cols,totrow,desfac;
    int i,j,k,l;
    int c;          /* # DIFFERENT ROWS */
    int gsfactor;   /* NUMBER OF TOTAL FACTORS FOR GROUP
SCREENING */
    int pb;
    char ch = 's';
    char wait_key;

    orthog = 1;
    choice = 0;
    pb = 0; /* TELLS WHETHER OR NOT PLACKETT-BURMAN DESIGN */

    if (gpscreen != 0)
    {
        gsfactor = factor; /* NUMBER OF TOTAL FACTORS FOR THE
GROUPS */
        factor = gpscreen; /* ACTUAL NUMBER OF FACTORS FOR
GROUP SCREENING */
    }
}

```

```

/* GENERATE 2 LEVEL [1-,1] DESIGN MATRIX */
if (level == 2)
{
    if (factor == 2)
    /* 2**2 full factorial */
    { desfac = 2;
      row = pow(2,desfac)*(reps+1);
      cols = pow(2,desfac);
      totrow = row + ceps*(reps+1);
      design(desfac,row,ceps,totrow);
      lev2_label(desfac);
    }

    else if (factor == 3)
    { if (res == 3)
      /* 2**(3-1) res III design, 3=12 */
      { desfac = 2;
        row = pow(2,desfac)*(reps+1);
        cols = pow(2,desfac);
        totrow = row + ceps*(reps+1);
        design(desfac,row,ceps,totrow);
        lev2_label(desfac);
        desmat[0][4] = 3;
      }

      else
      /* 2**3 full factorial */
      { desfac = 3;
        cols = pow(2,desfac);
        row = pow(2,desfac)*(reps+1);
        totrow = row + ceps*(reps+1);
        design(desfac,row,ceps,totrow);
        lev2_label(desfac);
      }
    }

    else if (factor == 4)
    { if (res == 4)

```



```

/* 2**(4-1) res IV design, 4=123 */
{ desfac = 3;
  res = 4;
  cols = pow(2,desfac);
  row = pow(2,3)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
  desmat[0][8] = 4;
}
else

/* 2**4 full factorial */
{ desfac = 4;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
}
}

else if (factor == 5)

{ if (res == 3)

/* 2**(5-2) res III design, 4=12, 5=13 */
{ desfac = 3;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
  desmat[0][5] = 4;
  desmat[0][6] = 5;
}

else if (res == 5)

/* 2**(5-1) res V design, 5=1234 */
{ desfac = 4;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
  desmat[0][16] = 5;
}
}

```

```

    }
else
/* 2**5 full factorial */
{ desfac = 5;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
}

}

else if (factor == 6)
{ if (res == 3)
/* 2**(6-3) res III design, 4=12, 5=13, 6=23 */
{ desfac = 3;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
  desmat[0][5] = 4;
  desmat[0][6] = 5;
  desmat[0][7] = 6;
}

else if (res == 4)
/* 2**(6-2) res IV design, 5=123, 6=234 */
{ desfac = 4;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
  desmat[0][9] = 5;
  desmat[0][15] = 6;
}

else if (res == 5)
    printf("DESIGN NOT AVAILABLE\n");

else if (res == 6)

```

```

/* 2**(6-1) res VI design, 6=12345 */
{ desfac = 5;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
  desmat[0][32] = 6;
}

else

/* 2**6 full factorial */
{ desfac = 6;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
}

}

else if (factor == 7)

{ if (res == 3)

/* 2**(7-4) res III design, 4=12, 5=13, 6=23, 7=123
*/
{ desfac = 3;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);
  lev2_label(desfac);
  desmat[0][5] = 4;
  desmat[0][6] = 5;
  desmat[0][7] = 6;
  desmat[0][8] = 7;
}

else if (res == 4)

/* 2**(7-3) res IV design, 5=123, 6=234, 7=134 */
{ desfac = 4;
  cols = pow(2,desfac);
  row = pow(2,desfac)*(reps+1);
  totrow = row + ceps*(reps+1);
  design(desfac,row,ceps,totrow);

```

```

        lev2_label(desfac);
        desmat[0][9] = 5;
        desmat[0][15] = 6;
        desmat[0][14] = 7;
    }

    else if (res == 5)

        printf("DESIGN NOT AVAILABLE\n");

    else if (res == 6)

        printf("DESIGN NOT AVAILABLE\n");

    else if (res == 7)

        /* 2**(7-1) res VII design, 7=123456 */
        { desfac = 6;
          cols = pow(2,desfac);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
          desmat[0][64] = 6;
        }

    else printf("2**7 DESIGN UNAVAILABLE\n");

    /* 2**7 full factorial - UNAVAILABLE
    { desfac = 7;
      cols = pow(2,6);
      row = pow(2,desfac)*(reps+1);
      totrow = row + ceps*(reps+1);
      design(desfac,row,ceps,totrow);
      lev2_label(desfac);
    }
    */

}

else if (factor == 8)

{ if (res == 3)

    /* PLACKETT-BURMAN DESIGN */
    {
        row = 12;
        cols = row;
        orthog = 0;
    }
}

```

```

        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
    cols = 9;
    pb = 1;
    }

    else if (res == 4)

        /* 2**(8-4) res IV design, 5=234, 6=134, 7=123,
8=124 */
        { desfac = 4;
          cols = pow(2,desfac);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
          desmat[0][15] = 5;
          desmat[0][14] = 6;
          desmat[0][9] = 7;
          desmat[0][12] = 8;
        }

    else if (res == 5)

        /* 2**(8-2) res V design, 7=1234, 8=1256 */
        { desfac = 6;
          cols = pow(2,desfac);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
          desmat[0][18] = 7;
          desmat[0][52] = 8;
        }

    else if (res == 6 || res == 7)

        printf("DESIGN NOT AVAILABLE\n");

    else if (res == 8) printf("2**(8-1) DESIGN
UNAVAILABLE\n");

    /* 2**(8-1) res VIII design, 8=1234567
    { desfac = 7;
      cols = pow(2,6);
      row = pow(2,desfac)*(reps+1);
      totrow = row + ceps*(reps+1);
      design(desfac,row,ceps,totrow);
      lev2_label(desfac);

```

```

    }
    */
else
    printf("FULL FACTORIAL NOT AVAILABLE - 256 RUNS\n");
}

else if (factor == 9)
    { if (res == 3)
        /* 2**(9-5) res III design, 5=123, 6=234, 7=134,
8=124, 9=1234 */
        { desfac = 4;
          cols = pow(2,desfac);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
          desmat[0][9] = 5;
          desmat[0][15] = 6;
          desmat[0][14] = 7;
          desmat[0][12] = 8;
          desmat[0][16] = 9;
        }

        else if (res == 4)
            /* 2**(9-4) res IV design, 6=2345, 7=1345, 8=1245,
9=1235 */
            { desfac = 5;
              cols = pow(2,desfac);
              row = pow(2,desfac)*(reps+1);
              totrow = row + ceps*(reps+1);
              design(desfac,row,ceps,totrow);
              lev2_label(desfac);
              desmat[0][31] = 6;
              desmat[0][30] = 7;
              desmat[0][28] = 8;
              desmat[0][24] = 9;
            }

        else if (res == 5)
            printf("DESIGN NOT AVAILABLE\n");
    }

```

```

        else if (res == 6) printf("2**(9-2) DESIGN
UNAVAILABLE\n");

        /* 2**(9-2) res VI design, 8=13467 9=23567
        { desfac = 7;
          cols = pow(2,6);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
        } */

        else if (res == 7 || res == 8)

            printf("DESIGN NOT AVAILABLE\n");

        else

            printf("FULL FACTORIAL NOT AVAILABLE\n");

    }

    else if (factor == 10)

    { if (res == 3)

        /* 2**(10-6) res III design, 5=123, 6=234, 7=134,
            8=124, 9=1234, 10=12 */
        { desfac = 4;
          cols = pow(2,desfac);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
          desmat[0][9] = 5;
          desmat[0][15] = 6;
          desmat[0][14] = 7;
          desmat[0][12] = 8;
          desmat[0][16] = 9;
          desmat[0][6] = 10;
        }

        else if (res == 4)

            /* 2**(10-5) res IV design, 6=1234, 7=1235, 8=1245,
            9=1345
            10=2345 */

```

```

    { desfac = 5;
      cols = pow(2,desfac);
      row = pow(2,desfac)*(reps+1);
      totrow = row + ceps*(reps+1);
      design(desfac,row,ceps,totrow);
      lev2_label(desfac);
      desmat[0][17] = 6;
      desmat[0][24] = 7;
      desmat[0][28] = 8;
      desmat[0][30] = 9;
      desmat[0][31] = 10;
    }

    else if (res == 5)    printf("2**(10-3) DESIGN
UNAVAILABLE\n");

    /* 2**(10-3) res V design, 8=1237 9=2345 10=1346
    { desfac = 7;
      cols = pow(2,6);
      row = pow(2,desfac)*(reps+1);
      totrow = row + ceps*(reps+1);
      design(desfac,row,ceps,totrow);
      lev2_label(desfac);
    }
    */

    else

        printf("DESIGN NOT AVAILABLE\n");

    }

    else if (factor == 11)

        { if (res == 3)

            /* 2**(11-7) res III design, 5=123, 6=234, 7=134,
            8=124, 9=1234
                                                    10=12  11=13 */

            { desfac = 4;
              cols = pow(2,desfac);
              row = pow(2,desfac)*(reps+1);
              totrow = row + ceps*(reps+1);
              design(desfac,row,ceps,totrow);
              lev2_label(desfac);
              desmat[0][9] = 5;
              desmat[0][15] = 6;
              desmat[0][14] = 7;
              desmat[0][12] = 8;
              desmat[0][16] = 9;
            }
        }
    }

```



```

        desmat[0][6] = 10;
        desmat[0][7] = 11;
    }

    else if (res == 4)

        /* 2**(11-6) res IV design, 6=123, 7=234, 8=345,
9=134
                                10=145 11=245 */
        { desfac = 5;
          cols = pow(2,desfac);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
          desmat[0][10] = 6;
          desmat[0][16] = 7;
          desmat[0][29] = 8;
          desmat[0][15] = 9;
          desmat[0][26] = 10;
          desmat[0][27] = 11;
        }

        else if (res == 5)    printf("2**(11-4) DESIGN
UNAVAILABLE\n");

        /* 2**(11-4) res V design, 8=1237 9=2345 10=1346
11==1234567
        { desfac = 7;
          cols = pow(2,6);
          row = pow(2,desfac)*(reps+1);
          totrow = row + ceps*(reps+1);
          design(desfac,row,ceps,totrow);
          lev2_label(desfac);
        }
        */

        else    printf("DESIGN NOT AVAILABLE\n");

    }

    else if (factor == 12)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
        {row = 16;
          cols = row;
          orthog = 0;
          totrow = row + ceps*(reps+1);

```

```

        plackett(row,cols,ceps,totrow);
        cols = 13;
    }
}
else if (factor == 13)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 16;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 14;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
}
else if (factor == 14)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 16;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 15;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
}
else if (factor == 15)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 16;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 16;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
}
else if (factor == 16)
{
    if (res == 3)

```

```

        /* PLACKETT-BURMAN DESIGN */
    {
        row = 20;
        cols = row;
        orthog = 0;
        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
        cols = 17;}
    else printf("DESIGN NOT AVAILABLE\n");
}
else if (factor == 17)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
    {
        row = 20;
        cols = row;
        orthog = 0;
        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
        cols = 18;}
    else printf("DESIGN NOT AVAILABLE\n");
}
else if (factor == 18)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
    {
        row = 20;
        cols = row;
        orthog = 0;
        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
        cols = 19;}
    else printf("DESIGN NOT AVAILABLE\n");
}
else if (factor == 19)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
    {
        row = 20;
        cols = row;
        orthog = 0;
        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
        cols = 20;}
    }
}

```

```

        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 20)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 24;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 21;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 21)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 24;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 22;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 22)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 24;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 23;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 23)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {

```

```

        row = 24;
        cols = row;
        orthog = 0;
        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
        cols = 24;)
    else printf("DESIGN NOT AVAILABLE\n");
}
else if (factor == 24)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
        {
            row = 32;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 25;)
        else printf("DESIGN NOT AVAILABLE\n");
    }

else if (factor == 25)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
        {
            row = 32;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 26;)
        else printf("DESIGN NOT AVAILABLE\n");
    }

else if (factor == 26)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
        {
            row = 32;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 27;)
        else printf("DESIGN NOT AVAILABLE\n");
    }

```

```

    }
else if (factor == 27)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 32;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 28;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 28)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 32;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 29;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 29)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 32;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 30;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 30)
    { if (res == 3)
        /* PLACKETT-BURMAN DESIGN */
        {

```

```

        row = 32;
        cols = row;
        orthog = 0;
        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
        cols = 31;}
    else printf("DESIGN NOT AVAILABLE\n");
}
else if (factor == 31)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
        {
            row = 32;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 32;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 32)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
        {
            row = 36;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 33;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 33)

    { if (res == 3)

        /* PLACKETT-BURMAN DESIGN */
        {
            row = 36;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 34;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
}

```

```

else if (factor == 34)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 36;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 35;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 35)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 36;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 36;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 36)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 40;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 37;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
else if (factor == 37)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 40;
            cols = row;

```



```

        orthog = 0;
        totrow = row + ceps*(reps+1);
        plackett(row,cols,ceps,totrow);
        cols = 38;}
    else printf("DESIGN NOT AVAILABLE\n");
}
else if (factor == 38)
{
    if (res == 3)
    {
        /* PLACKETT-BURMAN DESIGN */
        {
            row = 40;
            cols = row;
            orthog = 0;
            totrow = row + ceps*(reps+1);
            plackett(row,cols,ceps,totrow);
            cols = 39;}
        else printf("DESIGN NOT AVAILABLE\n");
    }
    else if (factor == 39)
    {
        if (res == 3)
        {
            /* PLACKETT-BURMAN DESIGN */
            {
                row = 40;
                cols = row;
                orthog = 0;
                totrow = row + ceps*(reps+1);
                plackett(row,cols,ceps,totrow);
                cols = 40;}
            else printf("DESIGN NOT AVAILABLE\n");
        }
    }

    else printf("NO DESIGNS AVAILABLE FOR THAT NUMBER OF
FACTORS\n");
}

else printf("YOU MUST CHOOSE TWO FACTOR LEVELS\n");

if (factor >= 12) pb = 1;
/* IF PLACKETT-BURMAN DESIGNS THEN NO REPLICATION POSSIBLE
*/
if (pb == 1) {
    reps = 0;
    desfac = factor;
}
if (ceps == 0 && reps == 0) c = row;

```

```

if (ceps >= 1 && reps == 0) c = row + 1;
if (pb == 0 && ceps == 0 && reps >= 1) c = pow(2,desfac);
    else if (pb == 1 && ceps == 0 && reps >= 1) c = row;
if (pb == 0 && ceps >= 1 && reps >= 1) c = pow(2,desfac) +
1;
    else if (pb == 1 && ceps >= 1 && reps >= 1) c = row + 1;

/* SEND DESIGN INFORMATION TO FILE EXP.DES */
outfile = fopen("exp.des", "w");

fprintf(outfile, " %2d", gpscreen);
if (gpscreen != 0)
    fprintf(outfile, " %2d", gsfactor);
else
    fprintf(outfile, " %2d", factor);
fprintf(outfile, " %2d", totrow);
fprintf(outfile, " %2d", cols);
fprintf(outfile, " %2d", desfac);
fprintf(outfile, " %2d", orthog);
fprintf(outfile, " %2d", choice);
fprintf(outfile, " %2d", level);
fprintf(outfile, " %2d", ceps);
fprintf(outfile, " %2d", reps);
fprintf(outfile, " %2d", c);
fprintf(outfile, " %2d\n", row);

for (i=1; i<=cols; i++)
    fprintf(outfile, "%4g ", desmat[0][i]);
fprintf(outfile, "\n");

for (i=1; i<=totrow; i++) {
    for (j=1; j<=cols; j++)
        fprintf(outfile, "%4g ", desmat[i][j]);
    fprintf(outfile, "\n");
}
fclose(outfile);
printf("NUMBER OF GROUPS %d\n", gpscreen);
if (gpscreen != 0)
    printf("NUMBER OF <GROUP SCREENING> ACTUAL FACTORS
%d\n", gsfactor);
else
    printf("NUMBER OF ACTUAL FACTORS %d\n", factor);
printf("NUMBER OF DESIGN FACTORS %d\n", desfac);
printf("NUMBER OF ROWS IN CORE DESIGN %d\n", row);
printf("TOTAL NUMBER OF ROWS %d\n", totrow);
printf("NUMBER OF COLUMNS %3d\n", cols);
printf("NUMBER OF CENTER POINTS %3d\n", ceps);
printf("NUMBER OF OVERALL DESIGN REPS %3d\n", reps);

```

```

        printf("\n ***** \n PRESS RETURN TO
CONTINUE \n");
        wait_key = getchar();

/* PRINT OUT THE DESIGN MATRIX */

    printf("THE DESIGN MATRIX IS\n");
    printf("\n");

    for (i=0;i<=totrow;i++)
        { for (j=1;j<=(cols);j++)
            { printf("%4.0f",desmat[i][j]); }
          printf("%3d\n",i);
        }
    printf("\n");
    if (orthog == 0) printf("NON-ORTHOGONAL DESIGN\n");
    printf("\n ***** \n PRESS RETURN TO CONTINUE
\n");
    wait_key = getchar();
}
/* END OF PROGRAM MAIN */

```

```

/* SUBROUTINE DESIGN GENERATES THE DESIGN MATRIX AND
INTERACTIONS */
int design(facs,row,cep,totrow)

{ int i,j,k,l,m,p,col;

/* INITIALIZE # FACTORS, ROWS AND COLUMNS */

    col = facs + 1;

/* INITIATE THE DESIGN MATRIX WITH 1s */

    for (i=0;i<=totrow;i++)
        for (j=0;j<=64;j++)
            { desmat[i][j] = 1; }

    m = 1;
    p = 2;

/* CREATE CORE MAIN EFFECTS DESIGN MATRIX */

```

```

    for (j=2;j<=col;j++)
        { l=1;
          while(l<=row)
              { for (i=1;i<=(l-l+m);i++)
                  desmat[i][j] = -1;

                  l = l+p;
              }
          m = m*2;
          p = p*2;
        }

/* ADD CENTER POINTS, IF REQUESTED */

if (cep >= 1)
    {
        for (i=row+1; i<=totrow; i++)
            for (j=2; j<=64; j++)
                desmat[i][j] = 0;
    }

/* SUBROUTINE TO WRITE DESIGN MATRIX INTERACTIONS */
/* new terms: */

if (facs>=2)
    { for (i=1;i<=row;i++)
        desmat[i][col+1] = desmat[i][2]*desmat[i][3];
    }

if (facs>=3)
    { for (i=1;i<=row;i++)
        {desmat[i][col+2] = desmat[i][2]*desmat[i][4];
         desmat[i][col+3] = desmat[i][3]*desmat[i][4];
         desmat[i][col+4] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
        }
    }

if (facs>=4)
    { for (i=1;i<=row;i++)
        {desmat[i][col+5] = desmat[i][2]*desmat[i][5];
         desmat[i][col+6] = desmat[i][3]*desmat[i][5];
         desmat[i][col+7] =
desmat[i][2]*desmat[i][3]*desmat[i][5];
        }
    }

```

```

        desmat[i][col+8] = desmat[i][4]*desmat[i][5];
        desmat[i][col+9] =
desmat[i][2]*desmat[i][4]*desmat[i][5];
        desmat[i][col+10]=
desmat[i][3]*desmat[i][4]*desmat[i][5];
        desmat[i][col+11]=
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5];
    }
}

if (facs>=5)
{
    for (i=1;i<=row;i++)
    {
        desmat[i][col+12] = desmat[i][2]*desmat[i][6];
        desmat[i][col+13] = desmat[i][3]*desmat[i][6];
        desmat[i][col+14] =
desmat[i][2]*desmat[i][3]*desmat[i][6];
        desmat[i][col+15] = desmat[i][4]*desmat[i][6];
        desmat[i][col+16] =
desmat[i][2]*desmat[i][4]*desmat[i][6];
        desmat[i][col+17] =
desmat[i][3]*desmat[i][4]*desmat[i][6];
        desmat[i][col+18] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][6];
        desmat[i][col+19] = desmat[i][5]*desmat[i][6];
        desmat[i][col+20] =
desmat[i][2]*desmat[i][5]*desmat[i][6];
        desmat[i][col+21] =
desmat[i][3]*desmat[i][5]*desmat[i][6];
        desmat[i][col+22] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+23] =
desmat[i][4]*desmat[i][5]*desmat[i][6];
        desmat[i][col+24] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+25] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+26] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5]*desmat[i][6];
    }
}

```

```

if (facs>=6)
{
  for (i=1;i<=row;i++)
  {
    desmat[i][col+27] = desmat[i][2]*desmat[i][7];
    desmat[i][col+28] = desmat[i][3]*desmat[i][7];
    desmat[i][col+29] =
desmat[i][2]*desmat[i][3]*desmat[i][7];
    desmat[i][col+30] = desmat[i][4]*desmat[i][7];
    desmat[i][col+31] =
desmat[i][2]*desmat[i][4]*desmat[i][7];
    desmat[i][col+32]=
desmat[i][3]*desmat[i][4]*desmat[i][7];
    desmat[i][col+33]=
desmat[i][2]*desmat[i][3]*desmat[i][4]*
desmat[i][7];

    desmat[i][col+34] = desmat[i][5]*desmat[i][7];
    desmat[i][col+35] =
desmat[i][2]*desmat[i][5]*desmat[i][7];
    desmat[i][col+36] =
desmat[i][3]*desmat[i][5]*desmat[i][7];
    desmat[i][col+37] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
desmat[i][7];

    desmat[i][col+38] =
desmat[i][4]*desmat[i][5]*desmat[i][7];
    desmat[i][col+39] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
desmat[i][7];

    desmat[i][col+40] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
desmat[i][7];

    desmat[i][col+41] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
desmat[i][5]*desmat[i][7];

    desmat[i][col+42] = desmat[i][6]*desmat[i][7];
    desmat[i][col+43] =
desmat[i][2]*desmat[i][6]*desmat[i][7];
    desmat[i][col+44] =
desmat[i][3]*desmat[i][6]*desmat[i][7];
    desmat[i][col+45] =
desmat[i][2]*desmat[i][3]*desmat[i][6]*
desmat[i][7];

    desmat[i][col+46] =
desmat[i][4]*desmat[i][6]*desmat[i][7];
    desmat[i][col+47] =
desmat[i][2]*desmat[i][4]*desmat[i][6]*
desmat[i][7];
  }
}

```

```

        desmat[i][col+48] =
desmat[i][3]*desmat[i][4]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+49] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][6]*desmat[i][7];

        desmat[i][col+50] =
desmat[i][5]*desmat[i][6]*desmat[i][7];
        desmat[i][col+51] =
desmat[i][2]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+52] =
desmat[i][3]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+53] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][6]*desmat[i][7];
        desmat[i][col+54] =
desmat[i][4]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+55] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][6]*desmat[i][7];
        desmat[i][col+56] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][6]*desmat[i][7];
        desmat[i][col+57] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
desmat[i][5]*desmat[i][6]*desmat[i][7];

    }
}

}
/* END OF TWO LEVEL DESIGN */

/* FUNCTION PLACKETT GENERATES RESOLUTION III
   PLACKETT-BURMAN DESIGNS */

plackett(int row, int col, int cep, int totrow)
{
    int i,j,k;

    if (col == 8)
    {

```

```

    desmat[1][1] = 1;    desmat[1][2] = 1;        desmat[1][3]
= 1;
    desmat[1][4] = 1;    desmat[1][5] ==-1;        desmat[1][6]
= 1;
    desmat[1][7] ==-1;    desmat[1][8] ==-1;
}
else if (col == 12)
{
    desmat[1][1] = 1; desmat[1][2] = 1; desmat[1][3] = 1;
    desmat[1][4] ==-1; desmat[1][5] = 1; desmat[1][6] = 1;
    desmat[1][7] = 1; desmat[1][8] ==-1; desmat[1][9] ==-1;
    desmat[1][10] ==-1; desmat[1][11] = 1; desmat[1][12] ==-1;
}
else if (col == 16)
{
    desmat[1][1] = 1; desmat[1][2] = 1; desmat[1][3] = 1;
    desmat[1][4] = 1; desmat[1][5] = 1; desmat[1][6] ==-1;
    desmat[1][7] = 1; desmat[1][8] ==-1; desmat[1][9] = 1;
    desmat[1][10] = 1; desmat[1][11] ==-1; desmat[1][12] ==-1;
    desmat[1][13] = 1; desmat[1][14] ==-1; desmat[1][15] ==-1;
    desmat[1][16] ==-1;
}
else if (col == 20)
{
    desmat[1][1] = 1; desmat[1][2] = 1; desmat[1][3] = 1;
    desmat[1][4] ==-1; desmat[1][5] ==-1; desmat[1][6] = 1;
    desmat[1][7] = 1; desmat[1][8] = 1; desmat[1][9] = 1;
    desmat[1][10] ==-1; desmat[1][11] = 1; desmat[1][12] ==-1;
    desmat[1][13] = 1; desmat[1][14] ==-1; desmat[1][15] ==-1;
    desmat[1][16] ==-1; desmat[1][17] ==-1; desmat[1][18] = 1;
    desmat[1][19] = 1; desmat[1][20] ==-1;
}
else if (col == 24)
{
    desmat[1][1] = 1; desmat[1][2] = 1; desmat[1][3] = 1;
    desmat[1][4] = 1; desmat[1][5] = 1; desmat[1][6] = 1;
    desmat[1][7] ==-1; desmat[1][8] = 1; desmat[1][9] ==-1;
    desmat[1][10] = 1; desmat[1][11] = 1; desmat[1][12] ==-1;
    desmat[1][13] ==-1; desmat[1][14] = 1; desmat[1][15] = 1;
    desmat[1][16] ==-1; desmat[1][17] ==-1; desmat[1][18] = 1;
    desmat[1][19] ==-1; desmat[1][20] = 1; desmat[1][21] ==-1;
    desmat[1][22] ==-1; desmat[1][23] ==-1; desmat[1][24] ==-1;
}
else if (col == 32)
{
    desmat[1][1] = 1; desmat[1][2] ==-1; desmat[1][3] ==-1;
    desmat[1][4] ==-1; desmat[1][5] ==-1; desmat[1][6] = 1;
    desmat[1][7] ==-1; desmat[1][8] = 1; desmat[1][9] ==-1;
    desmat[1][10] = 1; desmat[1][11] = 1; desmat[1][12] = 1;
    desmat[1][13] ==-1; desmat[1][14] = 1; desmat[1][15] = 1;

```



```

        desmat[1][16] = -1; desmat[1][17] = -1; desmat[1][18] = -1;
        desmat[1][19] = 1; desmat[1][20] = 1; desmat[1][21] = 1;
        desmat[1][22] = 1; desmat[1][23] = 1; desmat[1][24] = -1;
        desmat[1][25] = -1; desmat[1][26] = 1; desmat[1][27] = 1;
        desmat[1][28] = -1; desmat[1][29] = 1; desmat[1][30] = -1;
        desmat[1][31] = -1; desmat[1][32] = 1;
    }
else if (col == 36)
{
    desmat[1][1] = 1; desmat[1][2] = -1; desmat[1][3] = 1;
    desmat[1][4] = -1; desmat[1][5] = 1; desmat[1][6] = 1;
    desmat[1][7] = 1; desmat[1][8] = -1; desmat[1][9] = -1;
    desmat[1][10] = -1; desmat[1][11] = 1; desmat[1][12] = 1;
    desmat[1][13] = 1; desmat[1][14] = 1; desmat[1][15] = 1;
    desmat[1][16] = -1; desmat[1][17] = 1; desmat[1][18] = 1;
    desmat[1][19] = 1; desmat[1][20] = -1; desmat[1][21] = -1;
    desmat[1][22] = 1; desmat[1][23] = -1; desmat[1][24] = -1;
    desmat[1][25] = -1; desmat[1][26] = -1; desmat[1][27] = 1;
    desmat[1][28] = -1; desmat[1][29] = 1; desmat[1][30] = -1;
    desmat[1][31] = 1; desmat[1][32] = 1; desmat[1][33] = -1;
    desmat[1][34] = -1; desmat[1][35] = 1; desmat[1][36] = -1;
}
else if (col == 40)
{
    desmat[1][1] = 1; desmat[1][2] = 1; desmat[1][3] = 1;
    desmat[1][4] = -1; desmat[1][5] = -1; desmat[1][6] = 1;
    desmat[1][7] = 1; desmat[1][8] = 1; desmat[1][9] = 1;
    desmat[1][10] = -1; desmat[1][11] = 1; desmat[1][12] = -1;
    desmat[1][13] = 1; desmat[1][14] = -1; desmat[1][15] = -1;
    desmat[1][16] = -1; desmat[1][17] = -1; desmat[1][18] = 1;
    desmat[1][19] = 1; desmat[1][20] = -1; desmat[1][21] = 1;
    desmat[1][22] = 1; desmat[1][23] = 1; desmat[1][24] = -1;
    desmat[1][25] = -1; desmat[1][26] = 1; desmat[1][27] = 1;
    desmat[1][28] = 1; desmat[1][29] = 1; desmat[1][30] = -1;
    desmat[1][31] = 1; desmat[1][32] = -1; desmat[1][33] = 1;
    desmat[1][34] = -1; desmat[1][35] = -1; desmat[1][36] = -1;
    desmat[1][37] = -1; desmat[1][38] = 1; desmat[1][39] = 1;
    desmat[1][40] = -1;
}
else printf("DESIGN UNAVAILABLE FOR THE NUMBER OF FACTORS
REQUESTED\n");

for (i=2; i<=row-1; i++)
{
    for (j=2; j<=col; j++)
    {
        k = j-1;
        if (k <= 1) k = col;
        desmat[i][j] = desmat[i-1][k];
    }
}

for (i=1; i<=totrow; i++)

```

```

        desmat[i][1] = 1;
for (j=2; j<=col; j++)
    desmat[row][j] = -1;

/* ADD CENTER POINTS, IF REQUESTED */
if (cep >= 1)
{
    for (i=row+1; i<=totrow; i++)
        for (j=2; j<=64; j++)
            desmat[i][j] = 0;
}

/* ADD LABELS TO COLUMNS */
for (j=1; j<=col; j++)
    desmat[0][j] = j-1;
}

/* FUNCTION lev2_label ASSIGNS LABELS TO MATRIX COLUMNS */
int lev2_label(facs)
{
    int i,j,k,l,m,p,col;
    col = facs + 1;
    for (j=1; j<=col; j++)
        desmat[0][j] = j-1;

    if (facs>=2)    desmat[0][col+1] = 12;

    if (facs>=3)
    {
        desmat[0][col+2] = 13;
        desmat[0][col+3] = 23;
        desmat[0][col+4] = 123;
    }

    if (facs>=4)
    {
        desmat[0][col+5] = 14;
        desmat[0][col+6] = 24;
        desmat[0][col+7] = 124;
        desmat[0][col+8] = 34;
        desmat[0][col+9] = 134;
        desmat[0][col+10] = 234;
    }
}

```

```

        desmat[0][col+11]= 1234;
    }

    if (facs>=5)

        {desmat[0][col+12] = 15;
        desmat[0][col+13] = 25;
        desmat[0][col+14] = 125;
        desmat[0][col+15] = 35;
        desmat[0][col+16] = 135;
        desmat[0][col+17] = 235;
        desmat[0][col+18] = 1235;
        desmat[0][col+19] = 45;
        desmat[0][col+20] = 145;
        desmat[0][col+21] = 245;
        desmat[0][col+22] = 1245;
        desmat[0][col+23] = 345;
        desmat[0][col+24] = 1345;
        desmat[0][col+25] = 2345;
        desmat[0][col+26] = 12345;
        }

    if (facs>=6)

        {desmat[0][col+27] = 16;
        desmat[0][col+28] = 26;
        desmat[0][col+29] = 126;
        desmat[0][col+30] = 36;
        desmat[0][col+31] = 136;
        desmat[0][col+32]= 236;
        desmat[0][col+33]= 1236;
        desmat[0][col+34] = 46;
        desmat[0][col+35] = 146;
        desmat[0][col+36] = 246;
        desmat[0][col+37] = 1246;
        desmat[0][col+38] = 346;
        desmat[0][col+39] = 1346;
        desmat[0][col+40] = 2346;
        desmat[0][col+41] = 12346;
        desmat[0][col+42] = 56;
        desmat[0][col+43] = 156;
        desmat[0][col+44] = 256;
        desmat[0][col+45] = 1256;
        desmat[0][col+46] = 356;
        desmat[0][col+47] = 1356;
        desmat[0][col+48] = 2356;
        desmat[0][col+49] = 12356;
        desmat[0][col+50] = 456;
        desmat[0][col+51] = 1456;
        desmat[0][col+52] = 2456;
        }

```

```

        desmat[0][col+53] = 12456;
        desmat[0][col+54] = 3456;
        desmat[0][col+55] = 13456;
        desmat[0][col+56] = 23456;
        desmat[0][col+57] = 123456;
    }
}

```

LEVEL3.C

```

/* THIS PROGRAM GENERATES 3 LEVEL DESIGN MATRICES */

# include <stdio.h>
# include <math.h>

extern float alpha,desmat[65][130];
extern int gpscreen,factor,level;
extern int res,choice,orthog;
extern int ceps,reprs;

/* MAIN DETERMINES # FACTORS, RESOLUTION, AND DESIGN */

cdesign3()
{
FILE *outfile;
int row,desfac,cols,totrow;
int i,j,k,l;
char wait_key;
int c;

/* GENERATE 3 LEVEL [1-0,,1] DESIGN MATRIX */

if (level == 3)
{
    if (factor == 2)
    {
        if (choice == 1)

            /* 3 level 2**2 full factorial */
            { desfac = 2;
              row = pow(3,desfac)*(reprs+1);
              totrow = row + ceps*(reprs+1);
              desthree(desfac,row,ceps,totrow,choice);
              label(desfac);
            }

        else if (choice == 2)

```

```

        /* 3 level 2**2 central composite design for 2
factors */
        { desfac = 2;
          row = pow(2,desfac)*(reps+1);
          totrow = row + (ceps+(desfac*2))*(reps+1);
          desccd(desfac,row,totrow,alpha,ceps);
          label(desfac);
        }

        else printf("DESIGN UNAVAILABLE\n");
    }

    else if (factor == 3)
    {
        if (choice == 1)

            /* 3 level 2**3 full factorial */
            { desfac = 3;
              row = pow(3,desfac)*(reps+1);
              totrow = row + ceps*(reps+1);
              desthree(desfac,row,ceps,totrow,choice);
              label(desfac);
            }

            else if (choice == 2)

                /* 3 level 2**3 central composite design for 3
factors */
                { desfac = 3;
                  row = pow(2,desfac)*(reps+1);
                  totrow = row + (ceps+(desfac*2))*(reps+1);
                  desccd(desfac,row,totrow,alpha,ceps);
                  label(desfac);
                }

                else if (choice == 3)

                    /* BOX-BEHNKEN 3 FACTOR DESIGN */
                    { desfac = 3;
                      row = 12;
                      cols = 10;
                      ceps = 3;
                      totrow = 15;
                      box3(desfac,totrow);
                      boxint(desfac,totrow,row,choice);
                      boxlabel(desfac);
                    }
    }

```

```

    }
}

else if (factor == 4)
{
    if (choice==1)
        /* 3 level 2**4 full factorial */
        {
            desfac = 4;
            row = pow(3,desfac)*(reps+1);
            totrow = row + ceps*(reps+1);
            box4(desfac,totrow);
            desthree(desfac,row,ceps,totrow,choice);
            label(desfac);
        }

    else if (choice == 2)
        /* 3 level 2**4 central composite design for 4
factors */
        {
            desfac = 4;
            row = pow(2,desfac)*(reps+1);
            totrow = row + (ceps+(desfac*2))*(reps+1);
            descdd(desfac,row,totrow,alpha,ceps);
            label(desfac);
        }

    else if (choice == 3)
        /* BOX-BEHNKEN 4 FACTOR DESIGN */
        {
            desfac = 4;
            row = 24;
            cols = 15;
            ceps = 3;
            totrow = 27;
            box4(desfac,totrow);
            boxint(desfac,totrow,row,choice);
            boxlabel(desfac);
        }

}

else if (factor == 5)
{
    if (choice==1)

```

```

        /* 3 level 2**5 full factorial */
        { printf("3**5 DESIGN UNAVAILABLE \n");
        }

    else if (choice == 2)

        /* 3 level 2**5 central composite design for 5
factors */
        { desfac = 5;
          row = pow(2,desfac)*(reps+1);
          totrow = row + (ceps+(desfac*2))*(reps+1);
          desccd(desfac,row,totrow,alpha,ceps);
          label(desfac);
        }

    else if (choice == 3)

        /* BOX-BEHNKEN 5 FACTOR DESIGN */
        { desfac = 5;
          row = 40;
          cols = 21;
          ceps = 6;
          totrow = 46;
          box5(desfac,totrow);
          boxint(desfac,totrow,row,choice);
          boxlabel(desfac);
        }

    }

    else if (factor == 6)
    {

        if (choice==1)

            /* 3 level 2**6 full factorial */
            { printf("3**6 DESIGN UNAVAILABLE \n");
            }

        else if (choice == 2)

            /* 3 level 2**6 central composite design for 5
factors */
            { desfac = 6;
              row = pow(2,desfac)*(reps+1);
              totrow = row + (ceps+(desfac*2))*(reps+1);
              desccd(desfac,row,totrow,alpha,ceps);
              label(desfac);
            }
    }

```

```

    }

    else if (choice == 3)

        /* BOX-BEHNKEN 6 FACTOR DESIGN */
        {
            desfac = 6;
            row = 48;
            cols = 28;
            ceps = 6;
            totrow = 54;
            box6(desfac,totrow);
            boxint(desfac,totrow,row,choice);
            boxlabel(desfac);
        }

    }

    else printf("THREE LEVEL DESIGN UNAVAILABLE >6
FACTORS\n");

    }

    else printf("YOU MUST CHOOSE THREE FACTOR LEVELS\n");

gpscreen = 0;
if (choice == 1 && ceps >= 1) orthog = 0;
if (choice == 3) orthog = 0;

if (choice == 1 || choice == 2) cols = pow(2,desfac) +
desfac;

if (choice == 1)
{
    if (ceps == 0) c = pow(3,desfac);
    else c = pow(3,desfac) + 1;
}
if (choice == 2) c = pow(2,desfac) + 2*desfac + 1;
if (choice == 3) c = row + 1;

/* SEND DESIGN INFORMATION TO EXP.DES */

outfile = fopen("exp.des", "w");

fprintf(outfile, " %2d",gpscreen);
fprintf(outfile, " %2d",factor);
fprintf(outfile, " %2d",totrow);
fprintf(outfile, " %2d",cols);

```



```

fprintf(outfile," %2d",desfac);
fprintf(outfile," %2d",orthog);
fprintf(outfile," %2d",choice);
fprintf(outfile," %2d",level);
fprintf(outfile," %2d",ceps);
fprintf(outfile," %2d",reps);
fprintf(outfile," %2d",c);
fprintf(outfile," %2d\n",row);

for (j=1;j<=cols;j++)
    fprintf(outfile,"%4g ",desmat[0][j]);
fprintf(outfile,"\n");

for (i=1;i<=totrow;i++) {
    for (j=1;j<=cols;j++)
        fprintf(outfile,"%4g ",desmat[i][j]);
    fprintf(outfile,"\n");
}
fclose(outfile); /* Close the outfile for writing */

/* PRINT OUT DESIGN INFORMATION */

printf("NUMBER OF ACTUAL FACTORS %3d\n",factor);
printf("NUMBER OF ROWS IN CORE DESIGN %3d\n",row);
printf("TOTAL NUMBER OF ROWS %3d\n",totrow);
printf("NUMBER OF COLUMNS %3d\n",cols);
printf("NUMBER OF CENTER POINTS %3d\n",ceps);
printf("NUMBER OF OVERALL DESIGN REPS %3d\n",reps);
printf("\n ***** \n PRESS RETURN TO
CONTINUE \n");
wait_key = getchar();

/* PRINT OUT THE DESIGN MATRIX */

printf("THE DESIGN MATRIX IS\n");
printf("\n");

if (choice==1 || choice==3)
{
    for (i=0;i<=totrow;i++)
        { for (j=1;j<=(cols);j++)
            { printf("%4.0f",desmat[i][j]); }
          printf("%3d\n",i);
        }
    printf("\n");
}

if (choice == 2)
{
    for (i=0;i<=totrow;i++)

```

```

        { for (j=1;j<=(cols);j++)
          { printf("%5.2f",desmat[i][j]); }
          printf("%3d\n",i);
        }
        printf("\n");
      }
      printf("\n ***** \n PRESS RETURN TO
CONTINUE \n");
      wait_key = getchar();

    }
    /*  END OF PROGRAM MAIN  */

```

```

/* SUBROUTINE DESCCD GENERATES THE CCD DESIGN MATRIX AND
INTERACTIONS */
descdd(facs, row, totrow, alph, cep)
int facs, row, totrow, cep;
float alph;

{
  int i,j,k,l,m,p,col;
  float diff;

/* INITIALIZE # FACTORS, ROWS AND COLUMNS */

  col = facs + 1;

/* INITIATE THE DESIGN MATRIX WITH 1s */

  for (i=0;i<=totrow;i++)
    for (j=0;j<=64;j++)
      { desmat[i][j] = 1; }

  m = 1;
  p = 2;

/* CREATE CORE MAIN EFFECTS DESIGN MATRIX */

  for (j=2;j<=col;j++)
    { l=1;
      while(l<=row)
        { for (i=1;i<=(l-l+m);i++)
            desmat[i][j] = -1;

          l = l+p;
        }
    }

```

```

        m = m*2;
        p = p*2;
    }

/* CREATE THE EXTRA CCD ROWS */

/* PUT ZEROS IN ROWS FOR CCD DESIGNS */
    for (k=(row+1);k<=totrow;k++)
        {
            for
(j=2;j<=pow(2,col);j++)
                                desmat[k][j] =
0;
        }

/* PUT ALPHAS IN ROWS FOR CCD DESIGNS */

    l = row+cep; /* LAST CENTER POINT ROW */
    p = 1;
    while (l < totrow)

        {k = 2;

            for (m=l+1;m<=l+(2*facs);m=m+2)

                { desmat[m][k]    = -(alph);
                  desmat[m+1][k] =  alph;

                    k = k +1;
                }

            l = row + p*cep + p*(2*facs) + cep;
/* INCREMENT WITH:  CENTER PTS    ALPHA ROWS */
            p = p+1;
        }

/* SUBROUTINE TO WRITE DESIGN MATRIX INTERACTIONS */

/* new terms: */

    if (facs == 2)

        /* ADD INTERACTION TERMS FOR CCD DESIGNS */

        { for (i=1;i<=totrow;i++)
          {desmat[i][col+1] = desmat[i][2]*desmat[i][2];
            desmat[i][col+2] = desmat[i][3]*desmat[i][3];
            desmat[i][col+3] = desmat[i][2]*desmat[i][3];
          }
        }

```

```

        for (i=1; i<=totrow; i++)
            for (j=col+1; j<=col+2; j++)
                { diff = (row + 2.*alph*alph)/totrow;
                  desmat[i][j] = desmat[i][j] - diff;
                }
    }

else if (facs == 3)

    /* ADD SQUARED TERMS FOR CCD DESIGNS */

    { for (i=1; i<=totrow; i++)
      { desmat[i][col+1] = desmat[i][2]*desmat[i][2];
        desmat[i][col+2] = desmat[i][3]*desmat[i][3];
        desmat[i][col+3] = desmat[i][4]*desmat[i][4];
        desmat[i][col+4] = desmat[i][2]*desmat[i][3];
        desmat[i][col+5] = desmat[i][2]*desmat[i][4];
        desmat[i][col+6] = desmat[i][3]*desmat[i][4];
        desmat[i][col+7] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
      }
      for (i=1; i<=totrow; i++)
          for (j=col+1; j<=col+3; j++)
              { diff = (row + 2.*alph*alph)/totrow;
                desmat[i][j] = desmat[i][j] - diff;
              }
    }

else if (facs == 4)

    /* ADD SQUARED TERMS FOR CCD DESIGNS */

    { for (i=1; i<=totrow; i++)
      { desmat[i][col+1] = desmat[i][2]*desmat[i][2];
        desmat[i][col+2] = desmat[i][3]*desmat[i][3];
        desmat[i][col+3] = desmat[i][4]*desmat[i][4];
        desmat[i][col+4] = desmat[i][5]*desmat[i][5];
        desmat[i][col+5] = desmat[i][2]*desmat[i][3];
        desmat[i][col+6] = desmat[i][2]*desmat[i][4];
        desmat[i][col+7] = desmat[i][3]*desmat[i][4];
        desmat[i][col+8] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
        desmat[i][col+9] = desmat[i][2]*desmat[i][5];
        desmat[i][col+10] = desmat[i][3]*desmat[i][5];
        desmat[i][col+11] =
desmat[i][2]*desmat[i][3]*desmat[i][5];
        desmat[i][col+12] = desmat[i][4]*desmat[i][5];
        desmat[i][col+13] =
desmat[i][2]*desmat[i][4]*desmat[i][5];
      }
    }

```

```

        desmat[i][col+14] =
desmat[i][3]*desmat[i][4]*desmat[i][5];
        desmat[i][col+15] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5];
    }
    for (i=1; i<=totrow; i++)
        for (j=col+1; j<=col+4; j++)
            { diff = (row + 2.*alph*alph)/totrow;
              desmat[i][j] = desmat[i][j] - diff;
            }
    }

else if (facs == 5)

    { for (i=1; i<=totrow; i++)
      { desmat[i][col+1] = desmat[i][2]*desmat[i][2];
        desmat[i][col+2] = desmat[i][3]*desmat[i][3];
        desmat[i][col+3] = desmat[i][4]*desmat[i][4];
        desmat[i][col+4] = desmat[i][5]*desmat[i][5];
        desmat[i][col+5] = desmat[i][6]*desmat[i][6];
        desmat[i][col+6] = desmat[i][2]*desmat[i][3];
        desmat[i][col+7] = desmat[i][2]*desmat[i][4];
        desmat[i][col+8] = desmat[i][3]*desmat[i][4];
        desmat[i][col+9] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
        desmat[i][col+10] = desmat[i][2]*desmat[i][5];
        desmat[i][col+11] = desmat[i][3]*desmat[i][5];
        desmat[i][col+12] =
desmat[i][2]*desmat[i][3]*desmat[i][5];
        desmat[i][col+13] = desmat[i][4]*desmat[i][5];
        desmat[i][col+14] =
desmat[i][2]*desmat[i][4]*desmat[i][5];
        desmat[i][col+15] =
desmat[i][3]*desmat[i][4]*desmat[i][5];
        desmat[i][col+16] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5];
        desmat[i][col+17] = desmat[i][2]*desmat[i][6];
        desmat[i][col+18] = desmat[i][3]*desmat[i][6];
        desmat[i][col+19] =
desmat[i][2]*desmat[i][3]*desmat[i][6];
        desmat[i][col+20] = desmat[i][4]*desmat[i][6];
        desmat[i][col+21] =
desmat[i][2]*desmat[i][4]*desmat[i][6];
        desmat[i][col+22] =
desmat[i][3]*desmat[i][4]*desmat[i][6];
        desmat[i][col+23] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][6];
      }
    }

```

```

        desmat[i][col+24] = desmat[i][5]*desmat[i][6];
        desmat[i][col+25] =
desmat[i][2]*desmat[i][5]*desmat[i][6];
        desmat[i][col+26] =
desmat[i][3]*desmat[i][5]*desmat[i][6];
        desmat[i][col+27] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+28] =
desmat[i][4]*desmat[i][5]*desmat[i][6];
        desmat[i][col+29] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+30] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+31] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5]*desmat[i][6];
    }
    for (i=1; i<=totrow; i++)
        for (j=col+1; j<=col+5; j++)
            { diff = (row + 2.*alph*alph)/totrow;
              desmat[i][j] = desmat[i][j] - diff;
            }
}

else if (facs == 6)
{
    for (i=1; i<=totrow; i++)
    {
        desmat[i][col+1] = desmat[i][2]*desmat[i][2];
        desmat[i][col+2] = desmat[i][3]*desmat[i][3];
        desmat[i][col+3] = desmat[i][4]*desmat[i][4];
        desmat[i][col+4] = desmat[i][5]*desmat[i][5];
        desmat[i][col+5] = desmat[i][6]*desmat[i][6];
        desmat[i][col+6] = desmat[i][7]*desmat[i][7];
        desmat[i][col+7] = desmat[i][2]*desmat[i][3];
        desmat[i][col+8] = desmat[i][2]*desmat[i][4];
        desmat[i][col+9] = desmat[i][3]*desmat[i][4];
        desmat[i][col+10] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
        desmat[i][col+11] = desmat[i][2]*desmat[i][5];
        desmat[i][col+12] = desmat[i][3]*desmat[i][5];
        desmat[i][col+13] =
desmat[i][2]*desmat[i][3]*desmat[i][5];
        desmat[i][col+14] = desmat[i][4]*desmat[i][5];
        desmat[i][col+15] =
desmat[i][2]*desmat[i][4]*desmat[i][5];
        desmat[i][col+16] =
desmat[i][3]*desmat[i][4]*desmat[i][5];
    }
}

```

```

        desmat[i][col+17] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5];
        desmat[i][col+18] = desmat[i][2]*desmat[i][6];
        desmat[i][col+19] = desmat[i][3]*desmat[i][6];
        desmat[i][col+20] =
desmat[i][2]*desmat[i][3]*desmat[i][6];
        desmat[i][col+21] = desmat[i][4]*desmat[i][6];
        desmat[i][col+22] =
desmat[i][2]*desmat[i][4]*desmat[i][6];
        desmat[i][col+23] =
desmat[i][3]*desmat[i][4]*desmat[i][6];
        desmat[i][col+24] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][6];
        desmat[i][col+25] = desmat[i][5]*desmat[i][6];
        desmat[i][col+26] =
desmat[i][2]*desmat[i][5]*desmat[i][6];
        desmat[i][col+27] =
desmat[i][3]*desmat[i][5]*desmat[i][6];
        desmat[i][col+28] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+29] =
desmat[i][4]*desmat[i][5]*desmat[i][6];
        desmat[i][col+30] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+31] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+32] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5]*desmat[i][6];
        desmat[i][col+33] = desmat[i][2]*desmat[i][7];
        desmat[i][col+34] = desmat[i][3]*desmat[i][7];
        desmat[i][col+35] =
desmat[i][2]*desmat[i][3]*desmat[i][7];
        desmat[i][col+36] = desmat[i][4]*desmat[i][7];
        desmat[i][col+37] =
desmat[i][2]*desmat[i][4]*desmat[i][7];
        desmat[i][col+38] =
desmat[i][3]*desmat[i][4]*desmat[i][7];
        desmat[i][col+39] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][7];

        desmat[i][col+40] = desmat[i][5]*desmat[i][7];
        desmat[i][col+41] =
desmat[i][2]*desmat[i][5]*desmat[i][7];

```

```

        desmat[i][col+42] =
desmat[i][3]*desmat[i][5]*desmat[i][7];
        desmat[i][col+43] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][7];
        desmat[i][col+44] =
desmat[i][4]*desmat[i][5]*desmat[i][7];
        desmat[i][col+45] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][7];
        desmat[i][col+46] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][7];
        desmat[i][col+47] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5]*desmat[i][7];

        desmat[i][col+48] = desmat[i][6]*desmat[i][7];
        desmat[i][col+49] =
desmat[i][2]*desmat[i][6]*desmat[i][7];
        desmat[i][col+50] =
desmat[i][3]*desmat[i][6]*desmat[i][7];
        desmat[i][col+51] =
desmat[i][2]*desmat[i][3]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+52] =
desmat[i][4]*desmat[i][6]*desmat[i][7];
        desmat[i][col+53] =
desmat[i][2]*desmat[i][4]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+54] =
desmat[i][3]*desmat[i][4]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+55] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][6]*desmat[i][7];

        desmat[i][col+56] =
desmat[i][5]*desmat[i][6]*desmat[i][7];
        desmat[i][col+57] =
desmat[i][2]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+58] =
desmat[i][3]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+59] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][6]*desmat[i][7];
        desmat[i][col+60] =
desmat[i][4]*desmat[i][5]*desmat[i][6]*

```



```

                                desmat[i][7];
        desmat[i][col+61] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
                                desmat[i][6]*desmat[i][7];
        desmat[i][col+62] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
                                desmat[i][6]*desmat[i][7];
        desmat[i][col+63] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
desmat[i][5]*desmat[i][6]*desmat[i][7];
    }
    for (i=1; i<=totrow; i++)
        for (j=col+1; j<=col+6; j++)
            { diff = (row + 2.*alph*alph)/totrow;
              desmat[i][j] = desmat[i][j] - diff;
            }
    }
}

/* SUBROUTINE DESTHREE GENERATES THREE LEVEL DESIGNS */
int desthree(facs,row,cep,totrow,choice)
{
    int i,j,k,l,m,p,col;

/* INITIALIZE # FACTORS, ROWS, AND COLUMNS */
    col = facs + 1;

/* INITIATE THE DESIGN MATRIX WITH 1s */
    if (choice == 1)
    {
        for (i=0;i<=totrow;i++)
            for (j=0;j<=64;j++)
                { desmat[i][j] = 1; }

/* CREATE FULL THREE LEVEL CORE MAIN EFFECTS DESIGN MATRIX
*/
        m = 1;
        p = 3;

        for (j=2;j<=col;j++)
            { l=1;
              while(l<=row)

```

```

        { for (i=1;i<=(1-1+m);i++)
            desmat[i][j] = -1;
            l = l+p;
        }
        m = m*3;
        p = p*3;
    }

    m = 1;
    p = 3;
    k = 0;

    for (j=2;j<=col;j++)
    { l=pow(3,k)+1;
      while(l<=row)
      { for (i=1;i<=(1-1+m);i++)
          desmat[i][j] = 0;
          l = l+p;
        }
        m = m*3;
        p = p*3;
        k = k+1;
    }

/* ADD CENTER POINTS, IF REQUESTED */

if (cep >= 1)
{
    for (i=row+1; i<=totrow; i++)
        for (j=2; j<=64; j++)
            desmat[i][j] = 0;
}

}

/* SUBROUTINE TO WRITE DESIGN MATRIX INTERACTIONS */

/* new terms: */

if (facs==2)
{ for (i=1;i<=row;i++)
    { desmat[i][col+1] = desmat[i][2]*desmat[i][2];
      desmat[i][col+2] = desmat[i][3]*desmat[i][3];
      desmat[i][col+3] = desmat[i][2]*desmat[i][3];
    }
    if (choice == 1)

```

```

        {for (i=1; i<=totrow; i++)
            for (j=col+1; j<=col+2; j++)
                desmat[i][j] = desmat[i][j] -
(2./3.*row)/totrow;
        }
    }

    if (facs==3)
    { for (i=1;i<=row;i++)
        {desmat[i][col+1] = desmat[i][2]*desmat[i][2];
          desmat[i][col+2] = desmat[i][3]*desmat[i][3];
          desmat[i][col+3] = desmat[i][4]*desmat[i][4];
          desmat[i][col+4] = desmat[i][2]*desmat[i][3];
          desmat[i][col+5] = desmat[i][2]*desmat[i][4];
          desmat[i][col+6] = desmat[i][3]*desmat[i][4];
          desmat[i][col+7] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
        }
        if (choice == 1)
        {for (i=1; i<=totrow; i++)
            for (j=col+1; j<=col+3; j++)
                desmat[i][j] = desmat[i][j] -
(2./3.*row)/totrow;
        }
    }

    if (facs==4)
    { for (i=1;i<=row;i++)
        {desmat[i][col+1] = desmat[i][2]*desmat[i][2];
          desmat[i][col+2] = desmat[i][3]*desmat[i][3];
          desmat[i][col+3] = desmat[i][4]*desmat[i][4];
          desmat[i][col+4] = desmat[i][5]*desmat[i][5];
          desmat[i][col+5] = desmat[i][2]*desmat[i][3];
          desmat[i][col+6] = desmat[i][2]*desmat[i][4];
          desmat[i][col+7] = desmat[i][3]*desmat[i][4];
          desmat[i][col+8] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
          desmat[i][col+9] = desmat[i][2]*desmat[i][5];
          desmat[i][col+10]= desmat[i][3]*desmat[i][5];
          desmat[i][col+11]=
desmat[i][2]*desmat[i][3]*desmat[i][5];
          desmat[i][col+12]= desmat[i][4]*desmat[i][5];
          desmat[i][col+13]=
desmat[i][2]*desmat[i][4]*desmat[i][5];
          desmat[i][col+14]=
desmat[i][3]*desmat[i][4]*desmat[i][5];
          desmat[i][col+15]=
desmat[i][2]*desmat[i][3]*desmat[i][4]*

```

```

                                desmat[i][5];
        }
        if (choice == 1)
        {for (i=1; i<=totrow; i++)
            for (j=col+1; j<=col+4; j++)
                desmat[i][j] = desmat[i][j] -
(2./3.*row)/totrow;
        }
    }

    if (facs==5)
    { for (i=1;i<=row;i++)
        {desmat[i][col+1] = desmat[i][2]*desmat[i][2];
          desmat[i][col+2] = desmat[i][3]*desmat[i][3];
          desmat[i][col+3] = desmat[i][4]*desmat[i][4];
          desmat[i][col+4] = desmat[i][5]*desmat[i][5];
          desmat[i][col+5] = desmat[i][6]*desmat[i][6];
          desmat[i][col+6] = desmat[i][2]*desmat[i][3];
          desmat[i][col+7] = desmat[i][2]*desmat[i][4];
          desmat[i][col+8] = desmat[i][3]*desmat[i][4];
          desmat[i][col+9] =
desmat[i][2]*desmat[i][3]*desmat[i][4];
          desmat[i][col+10]= desmat[i][2]*desmat[i][5];
          desmat[i][col+11]= desmat[i][3]*desmat[i][5];
          desmat[i][col+12]=
desmat[i][2]*desmat[i][3]*desmat[i][5];
          desmat[i][col+13]= desmat[i][4]*desmat[i][5];
          desmat[i][col+14]=
desmat[i][2]*desmat[i][4]*desmat[i][5];
          desmat[i][col+15]=
desmat[i][3]*desmat[i][4]*desmat[i][5];
          desmat[i][col+16]=
desmat[i][2]*desmat[i][3]*desmat[i][4]*
desmat[i][5];
          desmat[i][col+17] = desmat[i][2]*desmat[i][6];
          desmat[i][col+18] = desmat[i][3]*desmat[i][6];
          desmat[i][col+19] =
desmat[i][2]*desmat[i][3]*desmat[i][6];
          desmat[i][col+20] = desmat[i][4]*desmat[i][6];
          desmat[i][col+21] =
desmat[i][2]*desmat[i][4]*desmat[i][6];
          desmat[i][col+22] =
desmat[i][3]*desmat[i][4]*desmat[i][6];
          desmat[i][col+23] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
desmat[i][6];
          desmat[i][col+24] = desmat[i][5]*desmat[i][6];
          desmat[i][col+25] =
desmat[i][2]*desmat[i][5]*desmat[i][6];
        }
    }

```

```

        desmat[i][col+26] =
desmat[i][3]*desmat[i][5]*desmat[i][6];
        desmat[i][col+27] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+28] =
desmat[i][4]*desmat[i][5]*desmat[i][6];
        desmat[i][col+29] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+30] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][6];
        desmat[i][col+31] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5]*desmat[i][6];
    }
    if (choice == 1)
    {for (i=1; i<=totrow; i++)
        for (j=col+1; j<=col+5; j++)
            desmat[i][j] = desmat[i][j] -
(2./3.*row)/totrow;
    }
}

if (facs==6)
{ for (i=1;i<=row;i++)
    {desmat[i][col+1] = desmat[i][2]*desmat[i][2];
desmat[i][col+2] = desmat[i][3]*desmat[i][3];
desmat[i][col+3] = desmat[i][4]*desmat[i][4];
desmat[i][col+4] = desmat[i][5]*desmat[i][5];
desmat[i][col+5] = desmat[i][6]*desmat[i][6];
desmat[i][col+6] = desmat[i][7]*desmat[i][7];
desmat[i][col+7] = desmat[i][2]*desmat[i][3];
desmat[i][col+8] = desmat[i][2]*desmat[i][4];
desmat[i][col+9] = desmat[i][3]*desmat[i][4];
desmat[i][col+10]=
desmat[i][2]*desmat[i][3]*desmat[i][4];
desmat[i][col+11]= desmat[i][2]*desmat[i][5];
desmat[i][col+12]= desmat[i][3]*desmat[i][5];
desmat[i][col+13]=
desmat[i][2]*desmat[i][3]*desmat[i][5];
desmat[i][col+14]= desmat[i][4]*desmat[i][5];
desmat[i][col+15]=
desmat[i][2]*desmat[i][4]*desmat[i][5];
desmat[i][col+16]=
desmat[i][3]*desmat[i][4]*desmat[i][5];
desmat[i][col+17]=
desmat[i][2]*desmat[i][3]*desmat[i][4]*

```

```

                                desmat[i][5];
desmat[i][col+18] = desmat[i][2]*desmat[i][6];
desmat[i][col+19] = desmat[i][3]*desmat[i][6];
desmat[i][col+20] =
desmat[i][2]*desmat[i][3]*desmat[i][6];
desmat[i][col+21] = desmat[i][4]*desmat[i][6];
desmat[i][col+22] =
desmat[i][2]*desmat[i][4]*desmat[i][6];
desmat[i][col+23] =
desmat[i][3]*desmat[i][4]*desmat[i][6];
desmat[i][col+24] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
                                desmat[i][6];
desmat[i][col+25] = desmat[i][5]*desmat[i][6];
desmat[i][col+26] =
desmat[i][2]*desmat[i][5]*desmat[i][6];
desmat[i][col+27] =
desmat[i][3]*desmat[i][5]*desmat[i][6];
desmat[i][col+28] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
                                desmat[i][6];
desmat[i][col+29] =
desmat[i][4]*desmat[i][5]*desmat[i][6];
desmat[i][col+30] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
                                desmat[i][6];
desmat[i][col+31] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
                                desmat[i][6];
desmat[i][col+32] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
                                desmat[i][5]*desmat[i][6];
desmat[i][col+33] = desmat[i][2]*desmat[i][7];
desmat[i][col+34] = desmat[i][3]*desmat[i][7];
desmat[i][col+35] =
desmat[i][2]*desmat[i][3]*desmat[i][7];
desmat[i][col+36] = desmat[i][4]*desmat[i][7];
desmat[i][col+37] =
desmat[i][2]*desmat[i][4]*desmat[i][7];
desmat[i][col+38] =
desmat[i][3]*desmat[i][4]*desmat[i][7];
desmat[i][col+39] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
                                desmat[i][7];

desmat[i][col+40] = desmat[i][5]*desmat[i][7];
desmat[i][col+41] =
desmat[i][2]*desmat[i][5]*desmat[i][7];
desmat[i][col+42] =
desmat[i][3]*desmat[i][5]*desmat[i][7];

```

```

        desmat[i][col+43] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][7];
        desmat[i][col+44] =
desmat[i][4]*desmat[i][5]*desmat[i][7];
        desmat[i][col+45] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][7];
        desmat[i][col+46] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][7];
        desmat[i][col+47] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][5]*desmat[i][7];

        desmat[i][col+48] = desmat[i][6]*desmat[i][7];
        desmat[i][col+49] =
desmat[i][2]*desmat[i][6]*desmat[i][7];
        desmat[i][col+50] =
desmat[i][3]*desmat[i][6]*desmat[i][7];
        desmat[i][col+51] =
desmat[i][2]*desmat[i][3]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+52] =
desmat[i][4]*desmat[i][6]*desmat[i][7];
        desmat[i][col+53] =
desmat[i][2]*desmat[i][4]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+54] =
desmat[i][3]*desmat[i][4]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+55] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
        desmat[i][6]*desmat[i][7];

        desmat[i][col+56] =
desmat[i][5]*desmat[i][6]*desmat[i][7];
        desmat[i][col+57] =
desmat[i][2]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+58] =
desmat[i][3]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];
        desmat[i][col+59] =
desmat[i][2]*desmat[i][3]*desmat[i][5]*
        desmat[i][6]*desmat[i][7];
        desmat[i][col+60] =
desmat[i][4]*desmat[i][5]*desmat[i][6]*
        desmat[i][7];

```

```

        desmat[i][col+61] =
desmat[i][2]*desmat[i][4]*desmat[i][5]*
        desmat[i][6]*desmat[i][7];
        desmat[i][col+62] =
desmat[i][3]*desmat[i][4]*desmat[i][5]*
        desmat[i][6]*desmat[i][7];
        desmat[i][col+63] =
desmat[i][2]*desmat[i][3]*desmat[i][4]*
desmat[i][5]*desmat[i][6]*desmat[i][7];
    }
    if (choice == 1)
    {for (i=1; i<=totrow; i++)
        for (j=col+1; j<=col+6; j++)
            desmat[i][j] = desmat[i][j] -
(2./3.*row)/totrow;
    }
}

/* CREATE BOX-BEHNKEN INTERACTIONS */
int boxint(facs,totrow,row,choice)
{
int i, j, col;

/* ADD CENTER POINTS */

    for (i=row+1; i<=totrow; i++)
        for (j=2; j<=64; j++)
            desmat[i][j] = 0;

/* ADD DESIGN MATRIX INTERACTIONS */

/* new terms: */

    col = facs + 1;

    if (facs==3)
    { for (i=1;i<=row;i++)
        {desmat[i][col+1] = desmat[i][2]*desmat[i][2];
        desmat[i][col+2] = desmat[i][3]*desmat[i][3];
        desmat[i][col+3] = desmat[i][4]*desmat[i][4];
        desmat[i][col+4] = desmat[i][2]*desmat[i][3];
        desmat[i][col+5] = desmat[i][2]*desmat[i][4];
        desmat[i][col+6] = desmat[i][3]*desmat[i][4];
        }
        for (i=1; i<=totrow; i++)

```



```

        for (j=col+1; j<=col+3; j++)
            desmat[i][j] = desmat[i][j] - (8./15.);
    }

    if (facs==4)
    {
        for (i=1; i<=row; i++)
        {
            desmat[i][col+1] = desmat[i][2]*desmat[i][2];
            desmat[i][col+2] = desmat[i][3]*desmat[i][3];
            desmat[i][col+3] = desmat[i][4]*desmat[i][4];
            desmat[i][col+4] = desmat[i][5]*desmat[i][5];
            desmat[i][col+5] = desmat[i][2]*desmat[i][3];
            desmat[i][col+6] = desmat[i][2]*desmat[i][4];
            desmat[i][col+7] = desmat[i][3]*desmat[i][4];
            desmat[i][col+8] = desmat[i][2]*desmat[i][5];
            desmat[i][col+9] = desmat[i][3]*desmat[i][5];
            desmat[i][col+10] = desmat[i][4]*desmat[i][5];
        }
        for (i=1; i<=totrow; i++)
            for (j=col+1; j<=col+4; j++)
                desmat[i][j] = desmat[i][j] - (12./27.);
    }

    if (facs==5)
    {
        for (i=1; i<=row; i++)
        {
            desmat[i][col+1] = desmat[i][2]*desmat[i][2];
            desmat[i][col+2] = desmat[i][3]*desmat[i][3];
            desmat[i][col+3] = desmat[i][4]*desmat[i][4];
            desmat[i][col+4] = desmat[i][5]*desmat[i][5];
            desmat[i][col+5] = desmat[i][6]*desmat[i][6];
            desmat[i][col+6] = desmat[i][2]*desmat[i][3];
            desmat[i][col+7] = desmat[i][2]*desmat[i][4];
            desmat[i][col+8] = desmat[i][3]*desmat[i][4];
            desmat[i][col+9] = desmat[i][2]*desmat[i][5];
            desmat[i][col+10] = desmat[i][3]*desmat[i][5];
            desmat[i][col+11] = desmat[i][4]*desmat[i][5];
            desmat[i][col+12] = desmat[i][2]*desmat[i][6];
            desmat[i][col+13] = desmat[i][3]*desmat[i][6];
            desmat[i][col+14] = desmat[i][4]*desmat[i][6];
            desmat[i][col+15] = desmat[i][5]*desmat[i][6];
        }
        if (choice == 1)
        {
            for (i=1; i<=totrow; i++)
                for (j=col+1; j<=col+5; j++)
                    desmat[i][j] = desmat[i][j] -
(2./3.*row)/totrow;
        }
    }
}

```

```

if (facs==6)
{
  for (i=1;i<=row;i++)
  {
    desmat[i][col+1] = desmat[i][2]*desmat[i][2];
    desmat[i][col+2] = desmat[i][3]*desmat[i][3];
    desmat[i][col+3] = desmat[i][4]*desmat[i][4];
    desmat[i][col+4] = desmat[i][5]*desmat[i][5];
    desmat[i][col+5] = desmat[i][6]*desmat[i][6];
    desmat[i][col+6] = desmat[i][7]*desmat[i][7];
    desmat[i][col+7] = desmat[i][2]*desmat[i][3];
    desmat[i][col+8] = desmat[i][2]*desmat[i][4];
    desmat[i][col+9] = desmat[i][3]*desmat[i][4];
    desmat[i][col+10] = desmat[i][2]*desmat[i][5];
    desmat[i][col+11] = desmat[i][3]*desmat[i][5];
    desmat[i][col+12] = desmat[i][4]*desmat[i][5];
    desmat[i][col+13] = desmat[i][2]*desmat[i][6];
    desmat[i][col+14] = desmat[i][3]*desmat[i][6];
    desmat[i][col+15] = desmat[i][4]*desmat[i][6];
    desmat[i][col+16] = desmat[i][5]*desmat[i][6];
    desmat[i][col+17] = desmat[i][2]*desmat[i][7];
    desmat[i][col+18] = desmat[i][3]*desmat[i][7];
    desmat[i][col+19] = desmat[i][4]*desmat[i][7];
    desmat[i][col+20] = desmat[i][5]*desmat[i][7];
    desmat[i][col+21] = desmat[i][6]*desmat[i][7];
  }
  if (choice == 1)
  {
    for (i=1; i<=totrow; i++)
      for (j=col+1; j<=col+6; j++)
        desmat[i][j] = desmat[i][j] -
(2./3.*row)/totrow;
  }
}

/* CREATE BOX-BEHNKEN CORE DESIGN */
int box3(facs,totrow)
{
  int i, j, col;

  for (i=1; i<=totrow; i++)
    desmat[i][1] = 1;

  if (facs == 3)
  {
    desmat[1][2] = -1; desmat[1][3] = -1; desmat[1][4] =
0;
    desmat[2][2] = 1; desmat[2][3] = -1; desmat[2][4] =
0;
    desmat[3][2] = -1; desmat[3][3] = 1; desmat[3][4] =
0;
  }
}

```

```

        desmat[4][2] = 1; desmat[4][3] = 1; desmat[4][4] =
0;
        desmat[5][2] = -1; desmat[5][3] = 0; desmat[5][4]
=-1;
        desmat[6][2] = 1; desmat[6][3] = 0; desmat[6][4]
=-1;
        desmat[7][2] = -1; desmat[7][3] = 0; desmat[7][4] =
1;
        desmat[8][2] = 1; desmat[8][3] = 0; desmat[8][4] =
1;
        desmat[9][2] = 0; desmat[9][3] = -1; desmat[9][4]
=-1;
        desmat[10][2] = 0; desmat[10][3] = 1;
desmat[10][4] = -1;
        desmat[11][2] = 0; desmat[11][3] = -1; desmat[11][4] =
1;
        desmat[12][2] = 0; desmat[12][3] = 1; desmat[12][4] =
1;
        desmat[13][2] = 0; desmat[13][3] = 0; desmat[13][4] =
0;
        desmat[14][2] = 0; desmat[14][3] = 0; desmat[14][4] =
0;
        desmat[15][2] = 0; desmat[15][3] = 0; desmat[15][4] =
0;
    }
}
/* CREATE BOX-BEHNKEN CORE DESIGN */
int box4(facs, totrow)
{
    int i, j, col;

    for (i=1; i<=totrow; i++)
        desmat[i][1] = 1;

    if (facs == 4)
    { desmat[1][2] = -1; desmat[1][3] = -1; desmat[1][4] =
0;
        desmat[2][2] = 1; desmat[2][3] = -1; desmat[2][4] =
0;
        desmat[3][2] = -1; desmat[3][3] = 1; desmat[3][4] =
0;
        desmat[4][2] = 1; desmat[4][3] = 1; desmat[4][4] =
0;
        desmat[5][2] = 0; desmat[5][3] = 0; desmat[5][4]
=-1;
        desmat[6][2] = 0; desmat[6][3] = 0; desmat[6][4] =
1;
        desmat[7][2] = 0; desmat[7][3] = 0; desmat[7][4]
=-1;

```

```

        desmat[8][2] = 0; desmat[8][3] = 0; desmat[8][4] =
1;
        desmat[9][2] = -1; desmat[9][3] = 0; desmat[9][4] =
0;
        desmat[10][2] = 1; desmat[10][3] = 0; desmat[10][4] =
0;
        desmat[11][2] = -1; desmat[11][3] = 0; desmat[11][4] =
0;
        desmat[12][2] = 1; desmat[12][3] = 0; desmat[12][4] =
0;
        desmat[13][2] = 0; desmat[13][3] = -1;
desmat[13][4] = -1;
        desmat[14][2] = 0; desmat[14][3] = 1;
desmat[14][4] = -1;
        desmat[15][2] = 0; desmat[15][3] = -1; desmat[15][4] =
1;
        desmat[16][2] = 0; desmat[16][3] = 1; desmat[16][4] =
1;
        desmat[17][2] = -1; desmat[17][3] = 0;
desmat[17][4] = -1;
        desmat[18][2] = 1; desmat[18][3] = 0;
desmat[18][4] = -1;
        desmat[19][2] = -1; desmat[19][3] = 0; desmat[19][4] =
1;
        desmat[20][2] = 1; desmat[20][3] = 0; desmat[20][4] =
1;
        desmat[21][2] = 0; desmat[21][3] = -1; desmat[21][4] =
0;
        desmat[22][2] = 0; desmat[22][3] = 1; desmat[22][4] =
0;
        desmat[23][2] = 0; desmat[23][3] = -1; desmat[23][4] =
0;
        desmat[24][2] = 0; desmat[24][3] = 1; desmat[24][4] =
0;
        desmat[25][2] = 0; desmat[25][3] = 0; desmat[25][4] =
0;
        desmat[26][2] = 0; desmat[26][3] = 0; desmat[26][4] =
0;
        desmat[27][2] = 0; desmat[27][3] = 0; desmat[27][4] =
0;

        desmat[1][5] = 0;
        desmat[2][5] = 0;
        desmat[3][5] = 0;
        desmat[4][5] = 0;
        desmat[5][5] = -1;
        desmat[6][5] = -1;
        desmat[7][5] = 1;
        desmat[8][5] = 1;
        desmat[9][5] = -1;

```

```

        desmat[10][5]=-1;
        desmat[11][5]= 1;
        desmat[12][5]= 1;
        desmat[13][5]= 0;
        desmat[14][5]= 0;
        desmat[15][5]= 0;
        desmat[16][5]= 0;
        desmat[17][5]= 0;
        desmat[18][5]= 0;
        desmat[19][5]= 0;
        desmat[20][5]= 0;
        desmat[21][5]=-1;
        desmat[22][5]=-1;
        desmat[23][5]= 1;
        desmat[24][5]= 1;
        desmat[25][5]= 0;
        desmat[26][5]= 0;
        desmat[27][5]= 0;
    }
}

/* CREATE BOX-BEHNKEN CORE DESIGN */
int box5(facs,totrow)
{
    int i, j, col;

    for (i=1; i<=totrow; i++)
        desmat[i][1] = 1;

    if (facs == 5)
    {
        desmat[1][2] = -1; desmat[1][3] = -1; desmat[1][4] =
0;
        desmat[2][2] = 1; desmat[2][3] = -1; desmat[2][4] =
0;
        desmat[3][2] = -1; desmat[3][3] = 1; desmat[3][4] =
0;
        desmat[4][2] = 1; desmat[4][3] = 1; desmat[4][4] =
0;
        desmat[5][2] = 0; desmat[5][3] = 0; desmat[5][4]
=-1;
        desmat[6][2] = 0; desmat[6][3] = 0; desmat[6][4] =
1;
        desmat[7][2] = 0; desmat[7][3] = 0; desmat[7][4]
=-1;
        desmat[8][2] = 0; desmat[8][3] = 0; desmat[8][4] =
1;
        desmat[9][2] = 0; desmat[9][3] = -1; desmat[9][4] =
0;
        desmat[10][2]= 0; desmat[10][3]= 1; desmat[10][4]=
0;
    }
}

```

```

        desmat[11][2]= 0; desmat[11][3]=-1; desmat[11][4]=
0;
        desmat[12][2]= 0; desmat[12][3]= 1; desmat[12][4]=
0;
        desmat[13][2]=-1; desmat[13][3]= 0;
desmat[13][4]=-1;
        desmat[14][2]= 1; desmat[14][3]= 0;
desmat[14][4]=-1;
        desmat[15][2]=-1; desmat[15][3]= 0; desmat[15][4]=
1;
        desmat[16][2]= 1; desmat[16][3]= 0; desmat[16][4]=
1;
        desmat[17][2]= 0; desmat[17][3]= 0; desmat[17][4]=
0;
        desmat[18][2]= 0; desmat[18][3]= 0; desmat[18][4]=
0;
        desmat[19][2]= 0; desmat[19][3]= 0; desmat[19][4]=
0;
        desmat[20][2]= 0; desmat[20][3]= 0; desmat[20][4]=
0;
        desmat[21][2]= 0; desmat[21][3]=-1;
desmat[21][4]=-1;
        desmat[22][2]= 0; desmat[22][3]= 1;
desmat[22][4]=-1;
        desmat[23][2]= 0; desmat[23][3]=-1; desmat[23][4]=
1;
        desmat[24][2]= 0; desmat[24][3]= 1; desmat[24][4]=
1;
        desmat[25][2]=-1; desmat[25][3]= 0; desmat[25][4]=
0;
        desmat[26][2]= 1; desmat[26][3]= 0; desmat[26][4]=
0;
        desmat[27][2]=-1; desmat[27][3]= 0; desmat[27][4]=
0;
        desmat[28][2]= 1; desmat[28][3]= 0; desmat[28][4]=
0;
        desmat[29][2]= 0; desmat[29][3]= 0;
desmat[29][4]=-1;
        desmat[30][2]= 0; desmat[30][3]= 0; desmat[30][4]=
1;
        desmat[31][2]= 0; desmat[31][3]= 0;
desmat[31][4]=-1;
        desmat[32][2]= 0; desmat[32][3]= 0; desmat[32][4]=
1;
        desmat[33][2]=-1; desmat[33][3]= 0; desmat[33][4]=
0;
        desmat[34][2]= 1; desmat[34][3]= 0; desmat[34][4]=
0;
        desmat[35][2]=-1; desmat[35][3]= 0; desmat[35][4]=
0;

```

```

0;      desmat[36][2]= 1; desmat[36][3]= 0; desmat[36][4]=
0;      desmat[37][2]= 0; desmat[37][3]=-1; desmat[37][4]=
0;      desmat[38][2]= 0; desmat[38][3]= 1; desmat[38][4]=
0;      desmat[39][2]= 0; desmat[39][3]=-1; desmat[39][4]=
0;      desmat[40][2]= 0; desmat[40][3]= 1; desmat[40][4]=
0;      desmat[41][2]= 0; desmat[41][3]= 0; desmat[41][4]=
0;      desmat[42][2]= 0; desmat[42][3]= 0; desmat[42][4]=
0;      desmat[43][2]= 0; desmat[43][3]= 0; desmat[43][4]=
0;      desmat[44][2]= 0; desmat[44][3]= 0; desmat[44][4]=
0;      desmat[45][2]= 0; desmat[45][3]= 0; desmat[45][4]=
0;      desmat[46][2]= 0; desmat[46][3]= 0; desmat[46][4]=
0;

```

```

desmat[1][5] = 0; desmat[1][6] = 0;
desmat[2][5] = 0; desmat[2][6] = 0;
desmat[3][5] = 0; desmat[3][6] = 0;
desmat[4][5] = 0; desmat[4][6] = 0;
desmat[5][5] = -1; desmat[5][6] = 0;
desmat[6][5] = -1; desmat[6][6] = 0;
desmat[7][5] = 1; desmat[7][6] = 0;
desmat[8][5] = 1; desmat[8][6] = 0;
desmat[9][5] = 0; desmat[9][6] = -1;
desmat[10][5] = 0; desmat[10][6] = -1;
desmat[11][5] = 0; desmat[11][6] = 1;
desmat[12][5] = 0; desmat[12][6] = 1;
desmat[13][5] = 0; desmat[13][6] = 0;
desmat[14][5] = 0; desmat[14][6] = 0;
desmat[15][5] = 0; desmat[15][6] = 0;
desmat[16][5] = 0; desmat[16][6] = 0;
desmat[17][5] = -1; desmat[17][6] = -1;
desmat[18][5] = 1; desmat[18][6] = -1;
desmat[19][5] = -1; desmat[19][6] = 1;
desmat[20][5] = 1; desmat[20][6] = 1;
desmat[21][5] = 0; desmat[21][6] = 0;
desmat[22][5] = 0; desmat[22][6] = 0;
desmat[23][5] = 0; desmat[23][6] = 0;
desmat[24][5] = 0; desmat[24][6] = 0;
desmat[25][5] = -1; desmat[25][6] = 0;
desmat[26][5] = -1; desmat[26][6] = 0;
desmat[27][5] = 1; desmat[27][6] = 0;

```

```

        desmat[28][5]= 1; desmat[28][6]= 0;
        desmat[29][5]= 0; desmat[29][6]=-1;
        desmat[30][5]= 0; desmat[30][6]=-1;
        desmat[31][5]= 0; desmat[31][6]= 1;
        desmat[32][5]= 0; desmat[32][6]= 1;
        desmat[33][5]= 0; desmat[33][6]=-1;
        desmat[34][5]= 0; desmat[34][6]=-1;
        desmat[35][5]= 0; desmat[35][6]= 1;
        desmat[36][5]= 0; desmat[36][6]= 1;
        desmat[37][5]=-1; desmat[37][6]= 0;
        desmat[38][5]=-1; desmat[38][6]= 0;
        desmat[39][5]= 1; desmat[39][6]= 0;
        desmat[40][5]= 1; desmat[40][6]= 0;
        desmat[41][5]= 0; desmat[41][6]= 0;
        desmat[42][5]= 0; desmat[42][6]= 0;
        desmat[43][5]= 0; desmat[43][6]= 0;
        desmat[44][5]= 0; desmat[44][6]= 0;
        desmat[45][5]= 0; desmat[45][6]= 0;
        desmat[46][5]= 0; desmat[46][6]= 0;
    }
}

/* CREATE BOX-BEHNKEN CORE DESIGN */
int box6(facs,totrow)
{
    int i, j, col;

    for (i=1; i<=totrow; i++)
        desmat[i][1] = 1;

    if (facs == 6)
    { desmat[1][2] ==-1; desmat[1][3] ==-1; desmat[1][4] =
0;
        desmat[2][2] = 1; desmat[2][3] ==-1; desmat[2][4] =
0;
        desmat[3][2] ==-1; desmat[3][3] = 1; desmat[3][4] =
0;
        desmat[4][2] = 1; desmat[4][3] = 1; desmat[4][4] =
0;
        desmat[5][2] ==-1; desmat[5][3] ==-1; desmat[5][4] =
0;
        desmat[6][2] = 1; desmat[6][3] ==-1; desmat[6][4] =
0;
        desmat[7][2] ==-1; desmat[7][3] = 1; desmat[7][4] =
0;
        desmat[8][2] = 1; desmat[8][3] = 1; desmat[8][4] =
0;
        desmat[9][2] = 0; desmat[9][3] ==-1; desmat[9][4]
=-1;

```



```

        desmat[10][2]= 0; desmat[10][3]= 1;
desmat[10][4]=-1;
        desmat[11][2]= 0; desmat[11][3]=-1; desmat[11][4]=
1;
        desmat[12][2]= 0; desmat[12][3]= 1; desmat[12][4]=
1;
        desmat[13][2]= 0; desmat[13][3]=-1;
desmat[13][4]=-1;
        desmat[14][2]= 0; desmat[14][3]= 1;
desmat[14][4]=-1;
        desmat[15][2]= 0; desmat[15][3]=-1; desmat[15][4]=
1;
        desmat[16][2]= 0; desmat[16][3]= 1; desmat[16][4]=
1;
        desmat[17][2]= 0; desmat[17][3]= 0;
desmat[17][4]=-1;
        desmat[18][2]= 0; desmat[18][3]= 0; desmat[18][4]=
1;
        desmat[19][2]= 0; desmat[19][3]= 0;
desmat[19][4]=-1;
        desmat[20][2]= 0; desmat[20][3]= 0; desmat[20][4]=
1;
        desmat[21][2]= 0; desmat[21][3]= 0;
desmat[21][4]=-1;
        desmat[22][2]= 0; desmat[22][3]= 0; desmat[22][4]=
1;
        desmat[23][2]= 0; desmat[23][3]= 0;
desmat[23][4]=-1;
        desmat[24][2]= 0; desmat[24][3]= 0; desmat[24][4]=
1;
        desmat[25][2]=-1; desmat[25][3]= 0; desmat[25][4]=
0;
        desmat[26][2]= 1; desmat[26][3]= 0; desmat[26][4]=
0;
        desmat[27][2]=-1; desmat[27][3]= 0; desmat[27][4]=
0;
        desmat[28][2]= 1; desmat[28][3]= 0; desmat[28][4]=
0;
        desmat[29][2]=-1; desmat[29][3]= 0; desmat[29][4]=
0;
        desmat[30][2]= 1; desmat[30][3]= 0; desmat[30][4]=
0;
        desmat[31][2]=-1; desmat[31][3]= 0; desmat[31][4]=
0;
        desmat[32][2]= 1; desmat[32][3]= 0; desmat[32][4]=
0;
        desmat[33][2]= 0; desmat[33][3]=-1; desmat[33][4]=
0;
        desmat[34][2]= 0; desmat[34][3]= 1; desmat[34][4]=
0;

```

```

    desmat[35][2]= 0; desmat[35][3]==-1; desmat[35][4]=
0;
    desmat[36][2]= 0; desmat[36][3]= 1; desmat[36][4]=
0;
    desmat[37][2]= 0; desmat[37][3]==-1; desmat[37][4]=
0;
    desmat[38][2]= 0; desmat[38][3]= 1; desmat[38][4]=
0;
    desmat[39][2]= 0; desmat[39][3]==-1; desmat[39][4]=
0;
    desmat[40][2]= 0; desmat[40][3]= 1; desmat[40][4]=
0;
    desmat[41][2]==-1; desmat[41][3]= 0;
desmat[41][4]==-1;
    desmat[42][2]= 1; desmat[42][3]= 0;
desmat[42][4]==-1;
    desmat[43][2]==-1; desmat[43][3]= 0; desmat[43][4]=
1;
    desmat[44][2]= 1; desmat[44][3]= 0; desmat[44][4]=
1;
    desmat[45][2]==-1; desmat[45][3]= 0;
desmat[45][4]==-1;
    desmat[46][2]= 1; desmat[46][3]= 0;
desmat[46][4]==-1;
    desmat[47][2]==-1; desmat[47][3]= 0; desmat[47][4]=
1;
    desmat[48][2]= 1; desmat[48][3]= 0; desmat[48][4]=
1;
    desmat[49][2]= 0; desmat[49][3]= 0; desmat[49][4]=
0;
    desmat[50][2]= 0; desmat[50][3]= 0; desmat[50][4]=
0;
    desmat[51][2]= 0; desmat[51][3]= 0; desmat[51][4]=
0;
    desmat[52][2]= 0; desmat[52][3]= 0; desmat[52][4]=
0;
    desmat[53][2]= 0; desmat[53][3]= 0; desmat[53][4]=
0;
    desmat[54][2]= 0; desmat[54][3]= 0; desmat[54][4]=
0;

    desmat[1][5] ==-1; desmat[1][6] = 0; desmat[1][7] =
0;
    desmat[2][5] ==-1; desmat[2][6] = 0; desmat[2][7] =
0;
    desmat[3][5] ==-1; desmat[3][6] = 0; desmat[3][7] =
0;
    desmat[4][5] ==-1; desmat[4][6] = 0; desmat[4][7] =
0;

```

```

0;      desmat[5][5] = 1; desmat[5][6] = 0; desmat[5][7] =
0;      desmat[6][5] = 1; desmat[6][6] = 0; desmat[6][7] =
0;      desmat[7][5] = 1; desmat[7][6] = 0; desmat[7][7] =
0;      desmat[8][5] = 1; desmat[8][6] = 0; desmat[8][7] =
0;      desmat[9][5] = 0; desmat[9][6] = -1; desmat[9][7] =
0;      desmat[10][5] = 0; desmat[10][6] = -1; desmat[10][7] =
0;      desmat[11][5] = 0; desmat[11][6] = -1; desmat[11][7] =
0;      desmat[12][5] = 0; desmat[12][6] = -1; desmat[12][7] =
0;      desmat[13][5] = 0; desmat[13][6] = 1; desmat[13][7] =
0;      desmat[14][5] = 0; desmat[14][6] = 1; desmat[14][7] =
0;      desmat[15][5] = 0; desmat[15][6] = 1; desmat[15][7] =
0;      desmat[16][5] = 0; desmat[16][6] = 1; desmat[16][7] =
0;      desmat[17][5] = -1; desmat[17][6] = 0;
desmat[17][7] = -1;
      desmat[18][5] = -1; desmat[18][6] = 0;
desmat[18][7] = -1;
      desmat[19][5] = 1; desmat[19][6] = 0;
desmat[19][7] = -1;
      desmat[20][5] = 1; desmat[20][6] = 0;
desmat[20][7] = -1;
      desmat[21][5] = -1; desmat[21][6] = 0; desmat[21][7] =
1;      desmat[22][5] = -1; desmat[22][6] = 0; desmat[22][7] =
1;      desmat[23][5] = 1; desmat[23][6] = 0; desmat[23][7] =
1;      desmat[24][5] = 1; desmat[24][6] = 0; desmat[24][7] =
1;      desmat[25][5] = -1; desmat[25][6] = -1; desmat[25][7] =
0;      desmat[26][5] = -1; desmat[26][6] = -1; desmat[26][7] =
0;      desmat[27][5] = 1; desmat[27][6] = -1; desmat[27][7] =
0;      desmat[28][5] = 1; desmat[28][6] = -1; desmat[28][7] =
0;      desmat[29][5] = -1; desmat[29][6] = 1; desmat[29][7] =
0;

```

```

    desmat[30][5]=-1; desmat[30][6]= 1; desmat[30][7]=
0;
    desmat[31][5]= 1; desmat[31][6]= 1; desmat[31][7]=
0;
    desmat[32][5]= 1; desmat[32][6]= 1; desmat[32][7]=
0;
    desmat[33][5]= 0; desmat[33][6]=-1;
desmat[33][7]=-1;
    desmat[34][5]= 0; desmat[34][6]=-1;
desmat[34][7]=-1;
    desmat[35][5]= 0; desmat[35][6]= 1;
desmat[35][7]=-1;
    desmat[36][5]= 0; desmat[36][6]= 1;
desmat[36][7]=-1;
    desmat[37][5]= 0; desmat[37][6]=-1; desmat[37][7]=
1;
    desmat[38][5]= 0; desmat[38][6]=-1; desmat[38][7]=
1;
    desmat[39][5]= 0; desmat[39][6]= 1; desmat[39][7]=
1;
    desmat[40][5]= 0; desmat[40][6]= 1; desmat[40][7]=
1;
    desmat[41][5]= 0; desmat[41][6]= 0;
desmat[41][7]=-1;
    desmat[42][5]= 0; desmat[42][6]= 0;
desmat[42][7]=-1;
    desmat[43][5]= 0; desmat[43][6]= 0;
desmat[43][7]=-1;
    desmat[44][5]= 0; desmat[44][6]= 0;
desmat[44][7]=-1;
    desmat[45][5]= 0; desmat[45][6]= 0; desmat[45][7]=
1;
    desmat[46][5]= 0; desmat[46][6]= 0; desmat[46][7]=
1;
    desmat[47][5]= 0; desmat[47][6]= 0; desmat[47][7]=
1;
    desmat[48][5]= 0; desmat[48][6]= 0; desmat[48][7]=
1;
    desmat[49][5]= 0; desmat[49][6]= 0; desmat[49][7]=
0;
    desmat[50][5]= 0; desmat[50][6]= 0; desmat[50][7]=
0;
    desmat[51][5]= 0; desmat[51][6]= 0; desmat[51][7]=
0;
    desmat[52][5]= 0; desmat[52][6]= 0; desmat[52][7]=
0;
    desmat[53][5]= 0; desmat[53][6]= 0; desmat[53][7]=
0;
    desmat[54][5]= 0; desmat[54][6]= 0; desmat[54][7]=
0;

```

```

    }
}

/* FUNCTION LABEL ASSIGNS LABELS TO MATRIX COLUMNS */

int label(facs)
{
    int i,j,k,l,m,p,col,col2;

    col = facs + 1;
    col2 = 2*facs + 1;

    desmat[0][1] = 0;

    for (j=2; j<=col; j++)
    {
        desmat[0][j] = j-1;
        desmat[0][j+facs] = 11*(j-1);
    }

    if (facs>=2)    desmat[0][col2+1] = 12;

    if (facs>=3)
    {
        desmat[0][col2+2] = 13;
        desmat[0][col2+3] = 23;
        desmat[0][col2+4] = 123;
    }

    if (facs>=4)
    {
        desmat[0][col2+5] = 14;
        desmat[0][col2+6] = 24;
        desmat[0][col2+7] = 124;
        desmat[0][col2+8] = 34;
        desmat[0][col2+9] = 134;
        desmat[0][col2+10] = 234;
        desmat[0][col2+11] = 1234;
    }

    if (facs>=5)
    {
        desmat[0][col2+12] = 15;
        desmat[0][col2+13] = 25;
        desmat[0][col2+14] = 125;
        desmat[0][col2+15] = 35;
        desmat[0][col2+16] = 135;
        desmat[0][col2+17] = 235;
        desmat[0][col2+18] = 1235;
        desmat[0][col2+19] = 45;
    }
}

```

```

        desmat[0][col2+20] = 145;
        desmat[0][col2+21] = 245;
        desmat[0][col2+22] = 1245;
        desmat[0][col2+23] = 345;
        desmat[0][col2+24] = 1345;
        desmat[0][col2+25] = 2345;
        desmat[0][col2+26] = 12345;
    }

    if (facs>=6)

        {desmat[0][col2+27] = 16;
        desmat[0][col2+28] = 26;
        desmat[0][col2+29] = 126;
        desmat[0][col2+30] = 36;
        desmat[0][col2+31] = 136;
        desmat[0][col2+32] = 236;
        desmat[0][col2+33] = 1236;
        desmat[0][col2+34] = 46;
        desmat[0][col2+35] = 146;
        desmat[0][col2+36] = 246;
        desmat[0][col2+37] = 1246;
        desmat[0][col2+38] = 346;
        desmat[0][col2+39] = 1346;
        desmat[0][col2+40] = 2346;
        desmat[0][col2+41] = 12346;
        desmat[0][col2+42] = 56;
        desmat[0][col2+43] = 156;
        desmat[0][col2+44] = 256;
        desmat[0][col2+45] = 1256;
        desmat[0][col2+46] = 356;
        desmat[0][col2+47] = 1356;
        desmat[0][col2+48] = 2356;
        desmat[0][col2+49] = 12356;
        desmat[0][col2+50] = 456;
        desmat[0][col2+51] = 1456;
        desmat[0][col2+52] = 2456;
        desmat[0][col2+53] = 12456;
        desmat[0][col2+54] = 3456;
        desmat[0][col2+55] = 13456;
        desmat[0][col2+56] = 23456;
        desmat[0][col2+57] = 123456;
        }
    }

/* FUNCTION boxLABEL ASSIGNS LABELS TO BOX-BEHNKEN MATRIX
COLUMNS */

int boxlabel(facs)

```

```

{  int i,j,k,l,m,p,col,col2;

col = facs + 1;
col2 = 2*facs + 1;

desmat[0][1] = 0;

for (j=2; j<=col; j++)
{  desmat[0][j] = j-1;
    desmat[0][j+facs] = 11*(j-1);
}

if (facs>=2)    desmat[0][col2+1] = 12;

if (facs>=3)
{
    desmat[0][col2+2] = 13;
    desmat[0][col2+3] = 23;
}

if (facs>=4)
{
    desmat[0][col2+4] = 14;
    desmat[0][col2+5] = 24;
    desmat[0][col2+6] = 34;
}

if (facs>=5)
{
    desmat[0][col2+7] = 15;
    desmat[0][col2+8] = 25;
    desmat[0][col2+9] = 35;
    desmat[0][col2+10] = 45;
}

if (facs>=6)
{
    desmat[0][col2+11] = 16;
    desmat[0][col2+12] = 26;
    desmat[0][col2+13] = 36;
    desmat[0][col2+14] = 46;
    desmat[0][col2+15] = 56;
}
}

```

VARSEL.C

```

/* VARSEL.C  TO REDUCE THE FACTOR SPACE FOR REGRESSION */
/*

```

```

*/

# include <string.h>
# include <stdio.h>
# include <math.h>
# include <dos.h>
# include "nandef.h"
# include "extend.h"

CLIPPER cvarsel()
{

FILE *infile, *outfile;
int i,j,k;
int gpscreen, factors, m, n;
int facs, orthog, choice, level, creps, reps, c, row;
int removed;
char ch;
char des_filename[12];

static float var_label[40];
static float x[40][40];
static int choose[40];

strcpy(des_filename,_parc(1,1));
if ((infile = fopen(des_filename, "r")) == NULL) {
    printf("File %s could not be opened\n", des_filename);
    return;
}

fscanf(infile,"%d",&gpscreen);
fscanf(infile,"%d",&factors);
fscanf(infile,"%d",&m);
fscanf(infile,"%d",&n);
fscanf(infile,"%d",&facs);
fscanf(infile,"%d",&orthog);
fscanf(infile,"%d",&choice);
fscanf(infile,"%d",&level);
fscanf(infile,"%d",&creps);
fscanf(infile,"%d",&reps);
fscanf(infile,"%d",&c);
fscanf(infile,"%d\n",&row);

printf("\n THE FOLLOWING IS THE FACTOR LIST FOR THE DESIGN
SELECTED\n");
for (i=1; i<=n; i++) {
    fscanf(infile,"%g",&var_label[i]);
    printf(" X%-.0f ",var_label[i]);
}
fscanf(infile,"\n");

```



```

printf("\n");

for (i=1; i<=m; i++) {
    for (j=1; j<=n; j++)
        fscanf(infile,"%g",&x[i][j]);
    fscanf(infile,"\n");
}

fclose(infile);

/* CASE TO SELECT VARIABLES */

removed = 0;
for (i=1; i<=n; i++) {
    printf("\nREMOVE VARIABLE => X%-.0f <= FROM THE
MODEL?\n",var_label[i]);
    printf("<1 = YES> TO REMOVE OR <RETURN> TO RETAIN
FACTOR\n");
    ch = getch();
    if (ch == '1') {
        removed = 1;
        choose[i] = 1;
        printf("\n***** FACTOR X%-.0f REMOVED FROM MODEL
*****\n",var_label[i]);
    }
}

if (removed == 0) {
    printf("NO VARIABLES SELECTED FOR REMOVAL <RETURN>\n");
    ch = getch();
    return;
}

/* THIS LOOP ACTUALLY COMPRESSES THE MATRIX BASED ON THE
DELETIONS */
for (j=1; j<=n; j++) {
    if (choose[j] == 1) {
        for (k=j; k<=n-1; k++) {
            for (i=1; i<=m; i++) {
                x[i][k] = x[i][k+1];
                var_label[k] = var_label[k+1];
                choose[k] = choose[k+1];
            }
            n = n-1;
            j = j-1;
        }
    }
}

printf("\nTHE X MATRIX\n");

```

```

for (i=1; i<=n; i++) printf("%6.0f ",var_label[i]);
printf("\n");
for (i=1; i<=m; i++) {
    for (j=1; j<=n; j++) {
        printf("%6.1f ",x[i][j]);
    }
    printf("\n");
}

```

```

printf("\n***** PRESS RETURN
*****");
getch();

```

```

/* WRITE exp.des AS CURRENT DESIGN FILE */
outfile = fopen("exp.des", "w");

```

```

fprintf(outfile," %2d",gpscreen);
fprintf(outfile," %2d",factors);
fprintf(outfile," %2d",m);
fprintf(outfile," %2d",n);
fprintf(outfile," %2d",facs);
fprintf(outfile," %2d",orthog);
fprintf(outfile," %2d",choice);
fprintf(outfile," %2d",level);
fprintf(outfile," %2d",creps);
fprintf(outfile," %2d",reps);
fprintf(outfile," %2d",c);
fprintf(outfile," %2d\n",row);

```

```

for (i=1; i<=n; i++)
    fprintf(outfile,"%4g ",var_label[i]);
fprintf(outfile,"\n");

```

```

for (i=1; i<=m; i++) {
    for (j=1; j<=n; j++)
        fprintf(outfile,"%4g ",x[i][j]);
    fprintf(outfile,"\n");
}
fclose(outfile);

```

```

) /* END OF CVARSEL.C FUNCTION */

```

TFORM.C

```

/* THIS PROGRAM TRANSFORMS RESPONSE DATA */

```

```

# include <string.h>
# include <stdio.h>

```

```

# include <math.h>
# include "nandef.h"
# include "extend.h"
char transform_res_filename[12] = "trans.res";

CLIPPER ctrans()
{
FILE *infile, *outfile;
int i,j,k;
int choice,num_runs;
float min,alpha;
char res_filename[12];

float y[65],transform_y[65];
strcpy(res_filename,_parc(1,1));
choice = _parni(1,2);
alpha = _parnd(1,3);
min = 1;

/* READ IN THE Y VECTOR FROM THE CORRECT FILE NAME */
infile = fopen(res_filename,"r");
fscanf(infile,"%d\n",&num_runs);
for (i=1; i<=num_runs; i++) fscanf(infile,"%g\n",&y[i]);
fclose(infile);

/* THE FUNCTION TRANSFORM IS USED FOR TRANSFORMATIONS ON
   THE RESPONSES (Y) */

/* THE CALLING TERM: transform(y,m1,transf,alpha); */

if (choice == 1)
{
/* PICK A POWER TRANSFORMATION: Y**alpha */
for (i=1; i<=num_runs; i++)
transform_y[i] = pow(y[i],alpha);
}
else if (choice == 2)
{
/* NATURAL LOG */
for (i=1; i<=num_runs; i++)
if (y[i] < min) min = y[i];
if (min < 0.) printf("TRANSFORMATION NOT POSSIBLE WITH
NEGATIVE VALUES\n");
else
{
for (i=1; i<=num_runs; i++)
if (y[i] != 0) transform_y[i] = log(y[i]);
}
}
}

```

```

else if (choice == 3)
{
    /* LOG BASE 10 */
    for (i=1; i<=num_runs; i++)
        if (y[i] < min) min = y[i];
    if (min < 0.) printf("TRANSFORMATION NOT POSSIBLE WITH
NEGATIVE VALUES\n");
    else
    {
        for (i=1; i<=num_runs; i++)
            if (y[i] != 0) transform_y[i] = log10(y[i]);
    }
}
else if (choice == 4)
{
    /* ARCSIN SQRT(Y) */
    for (i=1; i<=num_runs; i++)
        if (y[i] < min) min = y[i];
    for (i=1; i<=num_runs; i++)
        if (y[i] < -1. || y[i] > 1.) min = -1;
    if (min < 0.)
        (printf("TRANSFORMATION NOT POSSIBLE WITH NEGATIVE
VALUES\n");
        printf("AND VALUES MUST BE BETWEEN -1 AND +1 FOR ARCSIN
CALCULATION\n");
    }
    else
    {
        for (i=1; i<=num_runs; i++)
            transform_y[i] = asin(sqrt(y[i]));
    }
}
else if (choice == 5)
{
    /* LOG [(1+Y)/(1-Y)] */
    for (i=1; i<=num_runs; i++)
        if (y[i] < -1 || y[i] > 1) min = -1;
    if (min < 0.) printf("THE RESPONSES MUST BE BETWEEN 0 AND
1\n");
    else
    {
        for (i=1; i<=num_runs; i++)
            transform_y[i] = log((1+y[i])/(1-y[i]));
    }
}
else if (choice == 6)
{
    /* INVERSE OF Y */
    for (i=1; i<=num_runs; i++)
        if (y[i] != 0) transform_y[i] = 1.0/y[i];
}

```

```

    }
else if (choice == 7)
{
    /* SQUARE ROOT OF Y */
    for (i=1; i<=num_runs; i++)
        if (y[i] < min) min = y[i];
    if (min < 0.) printf("TRANSFORMATION NOT POSSIBLE WITH
NEGATIVE VALUES\n");
    else
    {
        for (i=1; i<=num_runs; i++)
            transform_y[i] = sqrt(y[i]);
    }
}
else if (choice == 8)
{
    /* SQUARE OF Y */
    for (i=1; i<=num_runs; i++)
        transform_y[i] = y[i]*y[i];
}
else if (choice == 9)
{
    /* LN(B-Y) */
    for (i=1; i<=num_runs; i++)
        if ((alpha - y[i]) < min) min = alpha - y[i];
    if (min <= 0.) printf("TRANSFORMATION NOT POSSIBLE WITH
NEGATIVE VALUES\n");
    else
    {
        for (i=1; i<=num_runs; i++)
            if(alpha-y[i] != 0) transform_y[i] = log(alpha - y[i]);
    }
}

printf("\nTHE ORIGINAL Y AND TRANSFORMED Y VECTOR(s)\n");
for (i=1; i<=num_runs; i++)
    printf("%10.5f      %10.5f\n",y[i],transform_y[i]);
printf("\n***** PRESS RETURN
*****\n");
getch();

/* WRITE THE TRANSFORMED RESPONSE VECTOR TO FILE */
outfile = fopen(transform_res_filename,"w");
fprintf(outfile,"%d\n",num_runs);
for (i=1; i<=num_runs; i++)
    fprintf(outfile,"%g\n",transform_y[i]);
fclose(outfile);
}

```

CLIPRAW

The following source file comprises the CLIPRAW executable module. This file builds the raw data matrix.

CLIPRAW.C

```
/* PROGRAM CLIPRAW.C -- A FUNCTION FOR CLIPPER */

# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# define ROW_LIM 130
# define COL_LIM 65
# define FAC_LIM 40
# define GRP_LIM 15

FILE *desfile,*infile,*outfile;
int return_value;
int i,j,k,l; /* COUNTERS */
int m,n; /* # ROWS AND # COLUMNS */
int c; /* # DIFFERENT LEVELS */
int orthog; /* ORTHOGONAL DESIGN FLAG <1> IS ORTHOGONAL */
int choice; /* CHOOSE THE DESIRED TRANSFORMATION */
static float low[ROW_LIM],high[ROW_LIM];
static float mid[ROW_LIM],semirange[ROW_LIM];
static float x[COL_LIM];
static int var_label[COL_LIM],design_col[FAC_LIM];
int level,row,facs,reprs,creps,gpscreen,factors,fac_factors;
char ch;

main()
{

/* READ FROM FILE EXP.DES */
desfile = fopen("exp.des","r");

fscanf(desfile,"%d",&gpscreen);
fscanf(desfile,"%d",&factors);
printf("GROUP %d FACTOR %d\n",gpscreen,factors);
fscanf(desfile,"%d",&m);
fscanf(desfile,"%d",&n);
fscanf(desfile,"%d",&facs);
fscanf(desfile,"%d",&orthog);
fscanf(desfile,"%d",&choice);
fscanf(desfile,"%d",&level);
fscanf(desfile,"%d",&creps);
fscanf(desfile,"%d",&reprs);
```

```

fscanf(desfile,"%d",&c);
fscanf(desfile,"%d\n",&row);

for (i=0; i<n; i++) {
    fscanf(desfile,"%d",&var_label[i]);
    if (var_label[i] > 0 && var_label[i] <= factors) {
        design_col[var_label[i]] = i ;
        printf("var_label = %d  design column =
%d\n",var_label[i],design_col[var_label[i]]);}
    }
fscanf(desfile,"\n");
/*
for (i=1; i<=m; i++) {
    for (j=0; j<n; j++)
        fscanf(infile,"%g",&x[i][j]);
    fscanf(infile,"\n");
} */

/* READ FROM FILE .FAC */
infile = fopen("exp.fac","r");

fscanf(infile,"%d",&fac_factors);
for (i=1; i<=factors; i++) {
    fscanf(infile,"%f %c %f",&low[i],&ch,&high[i]);
    while (ch != 10 && ch != EOF) ch = getc(infile);
    printf(" %.2f %.2f\n",low[i],high[i]); }

fclose(infile);

/* CREATE MID LEVEL & SEMIRANGE VALUES */
for (j=1; j<=factors; j++) {
    mid[j] = (high[j] + low[j])/2;
    semirange[j] = high[j] - mid[j];
}

outfile = fopen("exp.raw","w");

if (gpscreen > 0) {
    if (gpscreen > 15) {
        printf("\nPROGRAM LIMITATION -- ONLY 15 GROUPS ALLOWED
<RETURN>");
        getch();
        return(-15);
    }
    group_screening() ;
}
else { /* OTHER THAN GROUP RAW DATA CREATION */
    for (i=1; i<=m; i++) { /* FOR ALL OF THE ROWS */
        printf("%2d ",i);
        fprintf(outfile,"%2d ",i);
    }
}

```

```

/* READ IN A ROW FROM THE DESIGN FILE */
for (j=0; j<n; j++) fscanf(desfile,"%g",&x[j]);
for (j=1; j<=factors; j++) { /* FOR ALL OF THE FACTORS
*/
    if (x[design_col[j]] == -1) { /* LOW VALUE */
        printf("%f ",low[j]);
        fprintf(outfile,"%f ",low[j]);
    }
    else if (x[design_col[j]] == 1) { /* HIGH VALUE */
        printf("%f ",high[j]);
        fprintf(outfile,"%f ",high[j]);
    }
    else if (x[design_col[j]] == 0) { /* MIDDLE VALUE */
        printf("%f ",mid[j]);
        fprintf(outfile,"%f ",mid[j]);
    }
    else { /* ALPHA CCD VALUE */
        printf("%f ",(mid[j]+(semirange[j]*x[design_col[j]])));
        fprintf(outfile,"%f ",
            (mid[j]+(semirange[j]*x[design_col[j]])));
    } /* END OF IF STRUCTURE */
}
printf("\n");
fprintf(outfile,"\n"); /* ONLY ONE RUN PER OUTPUT LINE
*/
}
}
close(outfile);
printf("***** PRESS RETURN TO CONTINUE
*****\n");
ch = getchar();
close(desfile);
} /* END OF PROGRAM MAIN */

group_screening()
{
    /* GROUP DECLARATIONS */
    static int factor_groupnum[FAC_LIM];
    int group_table[FAC_LIM][GRP_LIM];
    int num_factors_ingroup[GRP_LIM];
    int factor_num;

    for (j=1; j<=GRP_LIM; j++) num_factors_ingroup[j] = 0;

    /* READ FROM FILE .GRP */
    infile = fopen("exp.grp","r");

    for (i=1; i<=factors; i++) {
        fscanf(infile,"%d",&factor_groupnum[i]);

```



```

    printf("FACTOR # %d IS IN GROUP #
%d\n",i,factor_groupnum[i]); }

fclose(infile);

for (i=1; i<=factors; i++) {
    k = factor_groupnum[i]; /* GIVES WHICH GROUP FACTOR IS
IN */
    num_factors_ingroup[k]++; /* COUNTER FOR THE # FACTORS
IN GROUP */
    l = num_factors_ingroup[k];
    /* PLACE THE FACTOR NUMBER IN THE GROUP NUMBER TABLE */
    group_table[l][k] = i;
}

for (i=1; i<=m; i++) { /* FOR ALL OF THE RUNS */
    printf("%2d ",i);
    fprintf(outfile,"%2d ",i);
    /* READ IN A ROW FROM THE DESIGN FILE */
    for (j=0; j<n; j++) fscanf(desfile,"%g",&x[j]);
    for (j=1; j<=gpscreen; j++) { /* FOR ALL OF THE FACTORS
*/
        for (k=1; k<=num_factors_ingroup[j]; k++) {
            factor_num = group_table[k][j];
            if (x[design_col[j]] == -1) { /* GROUP USES LOW
VALUE */
                printf("%.2f ",low[factor_num]);
                fprintf(outfile,"%2f ",low[factor_num]);
            }
            else { /* GROUP USES HIGH VALUE */
                printf("%.2f ",high[factor_num]);
                fprintf(outfile,"%2f ",high[factor_num]);
            }
        }
        printf(" "); /* PUT SPACES BETWEEN THE GROUPS */
        fprintf(outfile," "); /* PUT SPACES BETWEEN THE
GROUPS */
    }
    printf("\n"); /* ONLY ONE RUN PER OUTPUT LINE */
    fprintf(outfile,"\n"); /* ONLY ONE RUN PER OUTPUT LINE
*/
}
} /* END OF FUNCTION GROUP_SCREENING */

```

REGRESS

The following four C files comprise the REGRESS executable module. Regress1 is the driver for the executable module, while Regress2, Regress3, and Regress4 contain functions used during the regression calculations.

REGRESS1.C

```
/* REGRES1.C PERFORMS LINEAR REGRESSION WITH LEAST SQUARES
*/

# include <stdio.h>
# include <math.h>
# include <dos.h>
# include <string.h>

/* ZTBL IS A TABLE LOOKUP FOR THE RANKITS FUNCTION */
float ztbl[101] =
{
-2.500,-2.330,-2.055,-1.880,-1.750,-1.645,-1.555,-1.476,-1.4
05,-1.340,
-1.282,-1.226,-1.175,-1.126,-1.080,-1.037, -.995, -.954,
-.915, -.878,
-.842, -.807, -.773, -.738, -.707, -.675, -.643, -.613,
-.583, -.554,
-.525, -.496, -.468, -.440, -.412, -.385, -.358, -.332,
-.305, -.280,
-.253, -.227, -.202, -.176, -.150, -.126, -.100, -.075,
-.050, -.025,
.000, .025, .050, .075, .100, .126, .150, .176,
.202, .227,
.253, .280, .305, .332, .358, .385, .412, .440,
.468, .496,
.525, .554, .583, .613, .643, .675, .707, .738,
.773, .807,
.842, .878, .915, .954, .995, 1.037, 1.080, 1.126,
1.175, 1.226,
1.282, 1.340, 1.405, 1.476, 1.555, 1.645, 1.750, 1.880,
2.055, 2.330,
2.500};

/* MAIN CONTROLS THE REGRESSION PROGRAM */

main ()
{
```

```

FILE *infile, *outfile;
int i,j,k,l,m1,n1,n2,nn,p,q,r;      /* COUNTERS */
int m, n;                             /* # ROWS & COLUMNS */
int totdf, regdf, errdf;              /* DEGREES OF FREEDOM */
int sstocdf, ssrctf, ssecdf;          /* SUM OF SQUARES */
float ssr, ssto, sse;
float ssrct, sstoc, ssec;
float msr, msto, mse;
float msrct, msec;
float freg, fregp, rsq, rsqa;
float sum, ybar, mm;

static float x[35][35];               /* DESIGN MATRIX */
static float y[35][2];                /* RESPONSE VECTOR */
static float beta[35][2];             /* BETA ESTIMATES */
float betap[2][35];                  /* TRANSPOSED BETA */
float ss[2][2];                       /* SS scratch pad */
static float xpxinv[35][35];          /* INV(X'X) */

float f, t, alph, alpha;              /* CCD INV(X'X) CALCS */

static float stats[35][10];           /* REGRESSION STATISTICS */
static float xtrassr[35][2];          /* EXTRA SS ss */
static float ssrftest[35];

double fvalue();                      /* F TEST P-VALUE */
double tvalue();                      /* STUDENT T TEST P-VALUE */

/* LACK OF FIT VARIABLES */

float sspe, mspe;
float sslf, mslf;
float flof, flofp;                   /* F TEST, P-VALUE */
int level, row, row1;
int facs, totrow;
int reps, creps;                    /* # OF CENTER POINTS/REPS */
float reps1, creps1;                /* CONVERT TO FLOAT FOR CALS */
int c, sspef, sslfdf;
float ybart[35];

int gpscreen;
int factors;
int orthog;                          /* ORTHOGONAL DESIGN = 1 */
int choice;                          /* 2 LEVEL: RESOLUTION */
/* 3 LEVEL: FULL=1, CCD=2,

BOX-BEHNKEN=3 */
char ch= ',';                        /* VARIOUS USES */
char outname[24], output[10];        /* NAMES OF INPUT AND OUTPUT FILES */

```

```

/* READ FROM FILE EXP.DES */

if ((infile = fopen("exp.des", "r")) != NULL)
{
fscanf(infile,"%d",&gpscreen);
fscanf(infile,"%d",&factors);
fscanf(infile,"%d",&m);
fscanf(infile,"%d",&n);
fscanf(infile,"%d",&facs);
fscanf(infile,"%d",&orthog);
fscanf(infile,"%d",&choice);
fscanf(infile,"%d",&level);
fscanf(infile,"%d",&creps);
fscanf(infile,"%d",&reps);
fscanf(infile,"%d",&c);
fscanf(infile,"%d\n",&row);

for (i=1; i<=n; i++)
    fscanf(infile,"%g",&stats[i][5]);

for (i=1; i<=m; i++) {
    for (j=1; j<=n; j++)
        fscanf(infile,"%g",&x[i][j]);
    fscanf(infile,"\n");
}

fclose(infile);

/* READ RESPONSES FROM FILE */

if ( (infile = fopen("trans.res", "r")) ||
      (infile = fopen("exp.res", "r")) != NULL)
{
fscanf(infile,"%d\n",&m1);

for (i=1; i<=m1; i++)
    fscanf(infile,"%g",&y[i][1]);

fclose(infile);

/* BEGIN REGRESSION PROGRAM */
/* IDIOT CHECKS AND PROGRAM LIMITS */

if (m1 != m)

```

```

    { printf("\nNEED THE SAME NUMBER OF RESPONSES AS ROWS IN
DESIGN MATRIX <RETURN>\n");
      getch();
      return; }
if (n > m)
  { printf("\nCAN'T PERFORM REGRESSION WITH MORE COLUMNS
THAN ROWS <RETURN>\n");
    getch();
    return; }
if (m >= 34)
  { printf("\nPROGRAM IS LIMITED TO 34 ROWS <RETURN>\n");
    getch();
    return; }
if (n >= 34)
  { printf("\nPROGRAM IS LIMITED TO 34 COLUMNS
<RETURN>\n");
    getch();
    return; }
if (n < 2)
  { printf("\nMUST HAVE AT LEAST ONE VARIABLE IN MODEL
<RETURN>\n");
    getch();
    return; }
if (m < 4)
  { printf("\nMUST HAVE AT LEAST FOUR ROWS IN DESIGN MATRIX
<RETURN>\n");
    getch();
    return; }

creps = creps - 1;      /* ADJUST CEPS TO MATCH THIS PROGRAM
*/
creps1 = creps + 1.;    /* I.E., 1 REP = 2 CENTER POINTS */
reps1 = reps + 1.;      /* I.E., 1 R/P = 2 DESIGNS */

/* DEGREES OF FREEDOM */
ssrcdf = n-1;
ssecdf = m-n;
if (ssecdf == 0) ssecdf = 1;
sstocdf = m-1;

sspedf = m-c;
sslfd = c-n;
totrow = m;             /* # OF ROWS IN DESIGN MATRIX */

/* CALCULATE UNCORRECTED SUM OF SQUARES */

/* CALCULATE INV(X'X) */

```

```
xpxinvf(x,xpxinv,m,n,orthog,level,facs,row,choice,creps,rep
1);
```

```
/* CALCULATE BETAS */
```

```
betaf(xpxinv,x,y,beta,m,n);
```

```
/* CALCULATE Y-HAT STATISTICS */
```

```
yhatf(x,beta,stats,m,n);
```

```
/* CALCULATE UNCORRECTED SSR */
```

```
ssrf(beta, x, y, m, n, ss);
ssr = ss[1][1];
```

```
/* CALCULATE UNCORRECTED SSTO */
```

```
sstof(y,ss,m);
ssto = ss[1][1];
```

```
/* CALCULATE UNCORRECTED SSE */
```

```
sse = ssto - ssr;
if (sse <= .0001) sse = 1.;
```

```
/* CALCULATE UNCORRECTED MEAN SQUARED ERRORS
```

```
totdf = m;
regdf = n;
errdf = m-n;
msr = ssr/regdf;
msto = ssto/totdf;
mse = sse/errdf;          */
```

```
/* CALCULATION FOR SS CORRECTION */
```

```
sum = 0;
```

```
for (i=1; i<=m; i++)
    sum = sum + y[i][1];
```

```
/* CALCULATE CORRECTED SUMS OF SQUARES */
```

```

ssrc = ssr - (sum*sum)/m;

sstoc = ssto - (sum*sum)/m;

ssec = sse;

/* CALCULATE CORRECTED MEAN SQUARE ERRORS */

msrc = ssrc/ssrcdf;
msec = ssec/ssecdf;

/* MODEL STATISTICS: F TEST, P-VALUE, R SQUARED, AND ADJ R.
SQUARED */

freg = msrc/msec;
fregp = fvalue(freg,ssrcdf,ssecdf);
rsq = ssrc/sstoc;
rsqa = 1. - (sstocdf/ssecdf)*(ssec/sstoc);

/* COEFFICIENT STATISTICS: STANDARD ERRORS (stats[i][1]),
VARIANCES (stats[i][2]), T-STATISTICS (stats[i][3]),
AND P-VALUES (stats[i][4]) OF THE COEFFICIENTS */

for (i=1; i<=n; i++)
{ beta[i][0] = stats[i][5];
  stats[i][2] = xpxinv[i][i]*msec;
  stats[i][1] = sqrt(stats[i][2]);
  stats[i][3] = beta[i][1]/stats[i][1];
  stats[i][4] = tvalue(stats[i][3],ssecdf);
}

/* SORT THE COEFFICIENTS ON P-VALUES */

for (i=1; i<=n-1; i++)
{ k = i;
  for (nn=0; nn<=5; nn++)
    stats[0][nn] = stats[i][nn];
  for (j = i+1; j<=n; j++)
    { if (stats[j][4] < stats[0][4])
      { k = j;
        for (nn=0; nn<=5; nn++)
          stats[0][nn] = stats[j][nn];
      }
    }
  for (nn=0; nn<=5; nn++)
    { stats[k][nn] = stats[i][nn];
      stats[i][nn] = stats[0][nn];
    }
}

```

```

    }
}

/* CALCULATE EXTRA SUMS OF SQUARES */

nn = n;
for (i=1; i<=n; i++) xtrassr[i][0] = stats[i][5];

if (level == 2)
{
    if (orthog == 1) xby(x, beta, y, xtrassr, m, n);
    else
    {
        p = 1;
        extrassr(x, beta, y, xtrassr, m, p, nn, sum);
    }
}
else if (level == 3)
{
    if (orthog == 1) xby(x, beta, y, xtrassr, m, n);
    else
    {
        p = 1;
        extrassr(x, beta, y, xtrassr, m, p, nn, sum);
    }
}
else
{
    p = 1;
    extrassr(x, beta, y, xtrassr, m, p, nn, sum);
}

/* SORT THE EXTRA INDIVIDUAL SS FROM LARGEST ON DOWN
for (i=2; i<=n-1; i++)
{ k = i;
  for (nn=0; nn<=1; nn++)
    xtrassr[0][nn] = xtrassr[i][nn];
  for (j = i+1; j<=n; j++)
    { if (xtrassr[j][1] < xtrassr[0][1])
      { k = j;
        for (nn=0; nn<=1; nn++)
          xtrassr[0][nn] = xtrassr[j][nn];
      }
    }
  for (nn=0; nn<=1; nn++)
    { xtrassr[k][nn] = xtrassr[i][nn];

```



```

        xtrassr[i][nn] = xtrassr[0][nn];
    }
}
*/
/* CALCULATE F TEST ON THE EXTRA SS */
for (i=1; i<=n; i++) ssrftest[i] = xtrassr[i][1]/msec;

/* CALCULATE RANKITS(stats[i][9]) */
rankits(stats,m,ztbl);

/* CALCULATE LACK OF FIT */
/* FOR TWO LEVEL DESIGNS WITH CENTER POINTS AND/OR REPS */
for (i=1; i<=totrow; i++) ybart[i] = 0.;
sspe = 0.;

if (creps >= 1 || reps >= 1)
{
    if (level == 2)
    {
        if (creps < 1 && reps == 0)
            { printf("CANNOT COMPUTE LACK OF FIT WITHOUT REPS\n");
        }

        else if (creps < 0 && reps >= 1)
        {
            /* 1 -1 ROWS */
            row1 = pow(2,facs);
            k=1;
            while (k <= row1)
            {
                for (i=k; i<=totrow; i=i+row1)
                { ybart[k] = ybart[k] + y[i][1];
                }
                k = k+1;
            }

            for (k=1; k<=row1; k++)
            { ybart[k] = ybart[k] / reps1;
            }

            k=1;
            while (k<=row1)

```

```

        {
            for (i=k; i<=totrow; i=i+row1)
            {
                sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
            }
            k = k+1;
        }
    }

    else if (creps >= 1 && reps == 0)
    {
        /* CENTER POINTS */

        k = row + 1;
        for (i=(row+1); i<=(row+1+creps); i++)
        {
            ybart[k] = ybart[k] + y[i][1];
        }

        ybart[k] = ybart[k] / creps1;

        for (i=(row+1); i<=(row+1+creps); i++)
        {
            sspe = sspe + (y[i][1] - ybart[k])*(y[i][1] -
ybart[k]);
        }

    }

    else if (creps >= 0 && reps >= 1)
    {
        /* 1 -1 ROWS */
        row1 = pow(2,facs);
        k=1;
        while (k <= row1)
        {
            for (i=k; i<=row; i=i+row1)
            {
                ybart[k] = ybart[k] + y[i][1];
            }
            k = k+1;
        }

        for (k=1; k<=row1; k++)
        {
            ybart[k] = ybart[k] / reps1;
        }

        k=1;
    }

```

```

        while (k<=row1)
        {
            for (i=k; i<=row; i=i+row1)
            {
                sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
            }
            k = k+1;
        }

        /* CENTER POINTS */

        k = row + 1;
        for (i=row+1; i<=row+(creps1*reps1); i++)
        {
            ybart[k] = ybart[k] + y[i][1];
        }

        ybart[k] = ybart[k] / (creps1*reps1);

        for (i=row+1; i<=row+(creps1*reps1); i++)
        {
            sspe = sspe + (y[i][1] - ybart[k])*(y[i][1] -
ybart[k]);
        }
    }

}

/* LACK OF FIT FOR CCD DESIGNS */

if (level == 3 && choice == 2) /* choice = 2 = ccd
design */
{
    if (creps < 1 && reps == 0)
    { printf("CANNOT COMPUTE LACK OF FIT WITHOUT REPS\n");
    }

    else if (creps < 1 && reps >= 1)
    {
        /* 1 -1 ROWS */
        row1 = pow(2,facs);
        k=1;
        while (k <= row1)
        {
            for (i=k; i<=row1*(reps+1); i=i+row1)
            {
                ybart[k] = ybart[k] + y[i][1];
            }
        }
    }
}

```

```

        }
        k = k+1;
    }

    for (k=1; k<=row1; k++)
    {
        ybart[k] = ybart[k] / reps1;
    }

    k=1;
    while (k<=row1)
    {
        for (i=k; i<=row1*(reps+1); i=i+row1)
        {
            sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
        }
        k = k+1;
    }

    /* CENTER POINTS AND ALPHA ROWS */

    nn= (creps+1) + (2*facs);
    k = row + 1;
    while (k <= row + nn)
    {
        for (i=k; i<=totrow; i=i+nn)
        {
            ybart[k] = ybart[k] + y[i][1];
        }
        k = k+1;
    }

    for (k=row+1; k<=row+nn; k++)
    {
        ybart[k] = ybart[k] / reps1;
    }

    k=row+1;
    while (k<=row+nn)
    {
        for (i=k; i<=totrow; i=i+nn)
        {
            sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
        }
        k = k+1;
    }
}

```

```

else if (creps >= 1 && reps == 0)
{
    /* CENTER POINTS */

    k = row + 1;
    for (i=(row+1); i<=(row+1+creps); i++)
    {
        ybart[k] = ybart[k] + y[i][1];
    }

    ybart[k] = ybart[k] / creps1;

    for (i=(row+1); i<=(row+1+creps); i++)
    {
        sspe = sspe + (y[i][1] - ybart[k])*(y[i][1] -
ybart[k]);
    }

}

else if (creps >= 1 && reps >= 1)
{
    /* 1 -1 ROWS */
    row1 = pow(2,facs);
    k=1;
    while (k <= row1)
    {
        for (i=k; i<=row1*(reps+1); i=i+row1)
        {
            ybart[k] = ybart[k] + y[i][1];
        }
        k = k+1;
    }

    for (k=1; k<=row1; k++)
    {
        ybart[k] = ybart[k] / reps1;
    }

    k=1;
    while (k<=row1)
    {
        for (i=k; i<=row1*(reps+1); i=i+row1)
        {
            sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
        }
        k = k+1;
    }
}

```

```

/* CENTER POINTS */
nn= (creps+1) + (2*facs);
k = row + 1;
m1 = row+1;
while (m1 <= totrow)
{
    for (i=m1; i<=m1+creps; i++)
    {
        ybart[k] = ybart[k] + y[i][1];
    }
    m1 = m1+nn;
}

ybart[k] = ybart[k] / (reps1*creps1);

m1 = row+1;
while (m1 <= totrow)
{
    for (i=m1; i<=m1+creps; i++)
    {
        sspe = sspe + (y[i][1] - ybart[k])*(y[i][1] -
ybart[k]);
    }
    m1 = m1+nn;
}

/* ALPHA ROWS */

k = row + (creps+1) + 1;
while (k <= row + nn)
{
    for (i=k; i<=totrow; i=i+nn)
    {
        ybart[k] = ybart[k] + y[i][1];
    }
    k = k+1;
}

for (k=row+(creps+1)+1; k<=row+nn; k++)
{
    ybart[k] = ybart[k] / reps1;
}

k = row + (creps+1) + 1;
while (k<=row + nn)
{
    for (i=k; i<=totrow; i=i+nn)
    {
        sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
    }
}

```

```

        }
        k = k+1;
    }
}

/* THREE LEVEL DESIGNS */
if (level == 3 && (choice == 1 || choice == 3))
{
    if (creps < 1 && reps == 0)
    { printf("CANNOT COMPUTE LACK OF FIT WITHOUT REPS\n");
    }

    else if (creps < 0 && reps >= 1)
    {
        /* 1 -1 ROWS */
        row1 = pow(3,facs);
        k=1;
        while (k <= row1)
        {
            for (i=k; i<=totrow; i=i+row1)
            {
                ybart[k] = ybart[k] + y[i][1];
            }
            k = k+1;
        }

        for (k=1; k<=row1; k++)
        { ybart[k] = ybart[k] / reps1;
        }

        k=1;
        while (k<=row1)
        {
            for (i=k; i<=totrow; i=i+row1)
            {
                sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
            }
            k = k+1;
        }
    }

    else if (creps >= 1 && reps == 0)
    {
        /* CENTER POINTS */

        k = row + 1;
    }
}

```

```

        for (i=(row+1); i<=(row+1+creps); i++)
        {
            ybart[k] = ybart[k] + y[i][1];
        }

        ybart[k] = ybart[k] / creps1;

        for (i=(row+1); i<=(row+1+creps); i++)
        {
            sspe = sspe + (y[i][1] - ybart[k])*(y[i][1] -
ybart[k]);
        }

    }

    else if (creps >= 0 && reps >= 1)
    {
        /* 1 -1 ROWS */
        row1 = pow(3,facs);
        k=1;
        while (k <= row1)
        {
            for (i=k; i<=row; i=i+row1)
            {
                ybart[k] = ybart[k] + y[i][1];
            }
            k = k+1;
        }

        for (k=1; k<=row1; k++)
        {
            ybart[k] = ybart[k] / reps1;
        }

        k=1;
        while (k<=row1)
        {
            for (i=k; i<=row; i=i+row1)
            {
                sspe = sspe + (y[i][1] -
ybart[k])*(y[i][1] - ybart[k]);
            }
            k = k+1;
        }

        /* CENTER POINTS */

        k = row + 1;
        for (i=row+1; i<=totrow; i++)
        {

```



```

        ybart[k] = ybart[k] + y[i][1];
    }

    ybart[k] = ybart[k] / (creps1*reps1);

    for (i=row+1; i<=totrow; i++)
    {
        sspe = sspe + (y[i][1] - ybart[k])*(y[i][1] -
ybart[k]);
    }
}

/* LACK OF FIT STATISTICS */

mspe = sspe/sspedf;
sslf = ssec - sspe;
mslf = sslf/sslfd;
if (mspe <= .000001) mspe = .0001;
flof = mslf/mspe;
flop = fvalue(flof,sslfd,sspedf);
}

outfile = fopen("regress.out","w");

fprintf(outfile,"%d\n",m);
fprintf(outfile,"%d\n",n);
fprintf(outfile,"%d\n",facs);
fprintf(outfile,"%d\n",orthog);
fprintf(outfile,"%d\n",choice);
fprintf(outfile,"%d\n",level);
fprintf(outfile,"%d\n",creps);
fprintf(outfile,"%d\n",reps);
fprintf(outfile,"%d\n",c);
fprintf(outfile,"%d\n",row);

fprintf(outfile,"%g\n",ssrc);
fprintf(outfile,"%d\n",ssrcdf);
fprintf(outfile,"%g\n",msrc);
fprintf(outfile,"%g\n",freg);
fprintf(outfile,"%g\n",ssec);
fprintf(outfile,"%d\n",ssecdf);
fprintf(outfile,"%g\n",msec);
fprintf(outfile,"%g\n",sstoc);
fprintf(outfile,"%d\n",sstocdf);
fprintf(outfile,"%g\n",fregp);
fprintf(outfile,"%g\n",rsq);
fprintf(outfile,"%g\n",rsqa);

```

```

fprintf(outfile,"%g\n",sslf);
fprintf(outfile,"%d\n",sslfd);
fprintf(outfile,"%g\n",mslf);
fprintf(outfile,"%g\n",sspe);
fprintf(outfile,"%d\n",sspedf);
fprintf(outfile,"%g\n",mspe);
fprintf(outfile,"%g\n",flof);
fprintf(outfile,"%g\n",flofp);

for (i=1; i<=m; i++)
    for (j=1; j<=n; j++)
        fprintf(outfile,"%g %c",x[i][j],ch);

fprintf(outfile,"\n");

for (i=1; i<=m; i++)
    fprintf(outfile,"%g %c",y[i][1],ch);

fprintf(outfile,"\n");

for (i=1; i<=m; i++)
    for (j=0; j<=9; j++)
        fprintf(outfile,"%g %c",stats[i][j],ch);

fprintf(outfile,"\n");

for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        fprintf(outfile,"%g
%c",xpxinv[i][j],ch);

fprintf(outfile,"\n");

for (i=1; i<=m; i++)
    for (j=0; j<=1; j++)
        fprintf(outfile,"%g %c",beta[i][j],ch);

fprintf(outfile,"\n");

for (i=1; i<=m; i++)
    for (j=0; j<=2; j++)
        fprintf(outfile,"%g
%c",xtrassr[i][j],ch);

fprintf(outfile,"\n");

for (i=1; i<=m; i++)
    fprintf(outfile,"%g %c",ssrftest[i],ch);

fclose(outfile);

```

```

    }
    else
    { printf("\nCAN'T OPEN RESPONSE FILE <RETURN>\n");
      getch();
    }

  }
  else
  { printf("\nCAN'T OPEN DESIGN MATRIX FILE <RETURN>\n");
    getch();
  }
}

```

REGRESS2.C

```

/* FUNRES10.C SUPPORTS LINEAR REGRESSION WITH LEAST SQUARES
*/

# include <stdio.h>
# include <math.h>
# include <dos.h>
# include <string.h>

/* CALCULATE INV(X'X) */

xpxinvf(x,xpxinv,m,n,orthog,level,facs,row,choice,creps,rep
1)
float x[35][35], xpxinv[35][35], rep1;
int m, n, orthog, level, facs, row, choice, creps;
{
float scratch[35][35], scratchy[35][35], alph, f, t;
int i, j;

transpos(x,scratch,m,n);

matmulmm(scratch,x,scratchy,n,m,n);

if (orthog == 1)
{
if (level == 2 && facs < 12)
{ xpxinv[1][1] = 1./m;
for (i=2; i<=n; i++)
xpxinv[i][i] = 1./row;
}
else if (level == 3 && choice == 1)
invert(scratchy, xpxinv, n);
else if (level == 3 && choice == 2)

```

```

    {
        alph = x[row+creps+1+2][2];
        f = row;
        t = (2*facs+creps+1)*(reps1);
        xpxinv[1][1] = 1./m;
        for (i=2; i<= facs+1; i++)
            xpxinv[i][i] = 1./(row + 2*alph*alph);
        for (i=facs+2; i<=2*facs+1; i++)
            xpxinv[i][i] = 1./((f*t - 4*f*alph*alph -
4*pow(alph,4) +
                                2*(f+t)*pow(alph,4))/(f+t));
        for (i=2*facs+2; i<=n; i++)
            xpxinv[i][i] = 1./row;
    }
    else if (level == 3 && choice == 3)
    {
        invert(scratchy, xpxinv, n);
    }
}
else if (orthog == 0) invert(scratchy, xpxinv, n);
}

/* CALCULATE BETAS */

betaf(xpxinv,x,y,beta,m,n)
float xpxinv[35][35], x[35][35], y[35][2], beta[35][2];
int m, n;

{
float scratch[35][35], scratchy[35][35];
int i;

transpos(x, scratch, m, n);

matmulmm(xpxinv, scratch, scratchy, n, n, m);

matmulm1(scratchy, y, beta, n, m, 1);

}

/* CALCULATE Y-HAT STATISTICS */

yhatf(x,beta,stats,m,n)
float x[35][35], beta[35][2], stats[35][10];
int m, n;

{
float scratch3[35][2];
int i;

```

```

matmulm1(x,beta,scratch3,m,n,1);

for (i=1; i<=m; i++) stats[i][6] = scratch3[i][1];
for (i=1; i<=n; i++) stats[i][0] = beta[i][1];

)

/* CALCULATE UNCORRECTED SSR */

ssrf(beta, x, y, m, n, ss)
float beta[35][2], x[35][35], y[35][2], ss[2][2];
int m, n;
{
float betap[2][35], scratch[35][35], scratch1[2][35];

transpos2(beta, betap, n, 1);
transpos(x,scratch,m,n);
matmul1m(betap, scratch, scratch1, 1, n, m);
matmul11(scratch1, y, ss, 1, m, 1);
}

/* CALCULATE UNCORRECTED SSTO */

sstof(y,ss,m)
float y [35][2], ss[2][2];
int m;
{
float scratch1[2][35];

transpos2(y, scratch1, m, 1);
matmul11(scratch1, y, ss, 1, m, 1);
}

/* FUNCTION MATMULmm MULTIPLIES MATRIX a (mxn) BY MATRIX b
(nxm)
AND CREATES MATRIX c (mxm) */

```

```
matmulmm(float a[35][35], float b[35][35], float c[35][35],
        int m, int n, int p)
```

```
{
int i, j, k;
float sum;
```

```
    for (i=1; i<=m; i++)
```

```
        {for (j=1; j<=p; j++)
```

```
            {sum = 0.0;
```

```
                for (k=1; k<=n; k++)
```

```
                    { sum = sum + a[i][k] * b[k][j]; }
```

```
                c[i][j] = sum;
```

```
            }
```

```
        }
```

```
    }
```

```
/* FUNCTION MATMULnn MULTIPLIES MATRIX a (nx1) BY MATRIX b
(1xn)
```

```
AND CREATES MATRIX c (nxn) */
```

```
matmulnn(float a[35][2], float b[2][35], float c[35][35],
        int m, int n, int p)
```

```
{
int i, j, k;
float sum;
```

```
    for (i=1; i<=m; i++)
```

```
        {for (j=1; j<=p; j++)
```

```
            {sum = 0.0;
```

```
                for (k=1; k<=n; k++)
```

```
                    { sum = sum + a[i][k] * b[k][j]; }
```

```
                c[i][j] = sum;
```

```
            }
```

```
        }
```

```

}

/* FUNCTION MATMUL11 MULTIPLIES MATRIX a (1xn) BY MATRIX b
(nxl)
AND CREATES MATRIX c (1x1) */

matmul11(float a[2][35], float b[35][2], float c[2][2],
        int m, int n, int p)

{
    int i, j, k;
    float sum;

    for (i=1; i<=m; i++)
        (for (j=1; j<=p; j++)
            {sum = 0.0;
              for (k=1; k<=n; k++)
                  { sum = sum + a[i][k] * b[k][j]; }
              c[i][j] = sum;
            }
        )
}

/* FUNCTION MATMULm1 MULTIPLIES MATRIX a (mxn) BY MATRIX b
(nxl)
AND CREATES MATRIX c (mx1) */

matmulm1(float a[35][35], float b[35][2], float c[35][2],
        int m, int n, int p)

{
    int i, j, k;
    float sum;

    for (i=1; i<=m; i++)
        (for (j=1; j<=p; j++)
            {sum = 0.0;

```

```

        for (k=1; k<=n; k++)
            { sum = sum + a[i][k] * b[k][j]; }
        c[i][j] = sum;
    }
}

/* FUNCTION MATMUL1m MULTIPLIES MATRIX a (1xn) BY MATRIX b
(nxm)
AND CREATES MATRIX c (1xm) */
matmul1m(float a[2][35], float b[35][35], float c[2][35],
        int m, int n, int p)

{
    int i, j, k;
    float sum;

    for (i=1; i<=m; i++)
        {for (j=1; j<=p; j++)
            {sum = 0.0;
                for (k=1; k<=n; k++)
                    { sum = sum + a[i][k] * b[k][j]; }
                c[i][j] = sum;
            }
        }
}

/* FUNCTION transpos COMPUTES THE TRANPOSE OF A MATRIX a
(mxn)
AND CREATES MATRIX ap (nxm) */
transpos(float a[35][35], float ap[35][35], int m, int n)

{
    int i,j;

    for (i=0; i<=m; i++)

```



```

        for (j=0; j<=n; j++)
            ap[j][i] = a[i][j];
    }

/* FUNCTION transpos1 COMPUTES THE TRANPOSE OF A MATRIX a
(1xm)
AND CREATES MATRIX ap (mx1)    */
transpos1(float a[2][35], float ap[35][2], int m, int n)

{
    int i,j;

    for (i=0; i<=m; i++)
        for (j=0; j<=n; j++)
            ap[j][i] = a[i][j];
}

/* FUNCTION transpos2 COMPUTES THE TRANPOSE OF A MATRIX a
(mx1)
AND CREATES MATRIX ap (1xm)    */
transpos2(float a[35][2], float ap[2][35], int m, int n)

{
    int i,j;

    for (i=0; i<=m; i++)
        for (j=0; j<=n; j++)
            ap[j][i] = a[i][j];
}

REGRESS3.C

/* FUNRES11.C SUPPORTS LINEAR REGRESSION WITH LEAST SQUARES
*/

# include <stdio.h>
# include <math.h>
# include <dos.h>
# include <string.h>

/* THIS FUNCTION CALCULATES THE RANKITS    */
/* CALLING TERM: rankits(stats,m); */

```

```

rankits(stats,m,ztbl)

float stats[35][10],ztbl[101];
int m;
{
float rtmse, blom, blom100, remain, zvalue, newz, blomz;
int i, intblom, intblom1;

/* rtmse = sqrt(msec); */

for (i=1; i<=m; i++)
{ blom = (i-.375)/(m+.25);
  blom100 = blom*100;
  intblom = floor(blom100);
  remain = blom100 - intblom;
  intblom1 = intblom+1;
  zvalue = ztbl[intblom1] - ztbl[intblom];
  newz = zvalue*remain;
  blomz = ztbl[intblom] + newz;
  stats[i][9] = blomz;
}
}

/* FUNCTION GAMMLN IS USED TO CALCULATE P-VALUES */

double gammln(xx)
double xx;
{
/*
*****
*****
This is an incomplete gamma function pulled from Numerical
Recipes. It is
used to find T values and F values.
*****
***** */

/* VAR */
double x,tmp,ser;
int j;
double cof[6];
double stp, half, one, fpf;

/* CONST */
half = 0.5;
one = 1.0;
fpf = 5.5;

```

```

    cof[1] = 76.18009173;
    cof[2] = -86.50532033;
    cof[3] = 24.01409822;
    cof[4] = -1.231739516;
    cof[5] = 0.120858003e-2;
    cof[6] = -0.536382e-5;
    stp = 2.50662827465;
    x = xx-one;
    tmp = x+fpf;
    tmp = (x+half)*log(tmp)-tmp;
    ser = one;
    for (j = 1; j <= 6; j++)
        { x = x+one;
          ser = ser+cof[j]/x;
        }
    return (tmp+log(stp*ser));
}

```

```

double betacf(a,b,x)
double a, b, x;
{
    double    tem,qap,qam,qab,em,d;
    double    bz,bpp,bp,bm,az,app;
    double    am,aold,ap;
    int        m;
    int itmax;
    double eps;

    itmax=100;
    eps=3.0e-7;
    am = 1.0;
    bm = 1.0;
    az = 1.0;
    qab = a+b;
    qap = a+1.0;
    qam = a-1.0;
    bz = 1.0-qab*x/qap;
    for (m = 1; m <= itmax; m++)
        { em = m;
          tem = em+em;
          d = em*(b-m)*x/((qam+tem)*(a+tem));
          ap = az+d*am;
          bp = bz+d*bm;
          d = -(a+em)*(qab+em)*x/((a+tem)*(qap+tem));
          app = ap+d*az;
          bpp = bp+d*bz;
          aold = az;
          am = ap/bpp;

```

```

        bm = bp/bpp;
        az = app/bpp;
        bz = 1.0;
        if ((fabs(az-aold)) < (eps*fabs(az))) goto done;
    }
    printf("pause in BETACF\n");
    printf("a or b too big, or itmax too small\n");
done: return az;
}

double betai(a,b,x)
double a, b, x;
{
    /*
    *****
    *****
    This is an incomplete beta function used to generate T and Z
    values.  See
    Numerical Recipes.
    *****
    ***** */

    /* VAR */
    double bt;

    /* BEGIN */
    if (x < 0.0 || x > 1.0)
        printf("pause in routine BETAI");

    if (x == 0.0 || x == 1.0)
        bt = 0.0;
    else
        bt =
exp(gammln(a+b)-gammln(a)-gammln(b)+a*log(x)+b*log(1.0-x));

    if (x < ((a+1.0)/(a+b+2.0)))
        return (bt*betacf(a,b,x)/a);
    else
        return (1.0-bt*betacf(b,a,1.0-x)/b);
}

double tvalue(tstat,df)
double tstat;
int df;
{

```

```

/*
*****
*****
This is original, using a formula developed in Numerical
Recipes.
note: this returns the p value of t statistic and df, for
one
tailed test.
*****
***** */
double betai();

return betai(df/2.,1./2.,df/(df+tstat*tstat));
}

double fvalue(fstat,df1,df2)
double fstat;
int df1, df2;
{
/*
*****
*****
This returns the p value for fstat and the two degrees of
freedom.
*****
***** */
double betai();

return betai(df2/2.,df1/2.,df2/(df2+df1*fstat));
}

/* CALCULATE ORTHOGONAL EXTRA SUMS OF SQUARES */

xby(x, beta, y, xtrassr, m, n)
float x[35][35], beta[35][2], y[35][2], xtrassr[35][2];
int m, n;
{
float scratch[35][35], scratch1[2][35], scratch3[35][2];
int i;

    transpos(x,scratch,m,n);
    matmulm1(scratch,y,scratch3,n,m,1);
    transpos2(beta,scratch1,n,1);
    matmulnn(scratch3,scratch1,scratch,n,1,n);

    for (i=1; i<=n; i++)

```

```

        xtrassr[i][1] = scratch[i][i];
    }

/* CALCULATE NON-ORTHOGONAL EXTRA SUMS OF SQUARES */

extrassr(x, beta, y, xtrassr, m, n1, nn, sum)
float x[35][35], beta[35][2], y[35][2], xtrassr[35][2], sum;
int m, n1, nn;
{
    float scratch[35][35], scratchy[35][35], scratch1[2][35];
    float scratch2[35][35], scratch3[35][2], xpxinv[35][35];
    float ss[2][2], betap[2][35], cumssr[35];
    int i, j, q, r, n2;

    printf("\n\nPLEASE BE PATIENT. IT SOMETIMES TAKES A WHILE
    TO\n");
    printf("CALCULATE INDIVIDUAL SUMS OF SQUARES. \n");

    for (q=n1; q<=nn; q++)
    {
        for (r=1; r<=m; r++) scratch2[r][q] = x[r][q];

        /* CALCULATE BETAS */

        transpos(scratch2,scratch,m,q);

        matmulmm(scratch,scratch2,scratchy,q,m,q);

        invert(scratchy, xpxinv, q);

        matmulmm(xpxinv, scratch, scratchy, q, q, m);

        matmulm1(scratchy, y, beta, q, m, 1);

        /* CALCULATE CORRECTED EXTRA SSR */

        transpos2(beta, betap, q, 1);

        transpos(scratch2,scratch,m,q);

        matmul1m(betap, scratch, scratch1, 1, q, m);

        matmul11(scratch1, y, ss, 1, m, 1);

        cumssr[q] = ss[1][1] - (sum*sum)/m;

        if (q > 1) xtrassr[q][1] = cumssr[q] - cumssr[q-1];
    }
}

```

```
}  
}
```

REGRESS4.C

```
/* FUNRES12.C SUPPORTS LINEAR REGRESSION WITH LEAST SQUARES  
*/
```

```
# include <stdio.h>  
# include <math.h>  
# include <dos.h>  
# include <string.h>
```

```
/* THE FOLLOWING FUNCTIONS INVERT AN NXN MATRIX */
```

```
/* FUNCTION INVERT CONTROLS THE INVERSION PROCESS */
```

```
invert(float x[35][35], float ainv[35][35], int n)
```

```
{  
float a[35][35];  
float b[35];  
int ipivot[35];  
int i, j, k, iflag, ibeg;
```

```
iflag = 1;
```

```
for (i=1; i<=n; i++)  
    for (j=1; j<=n; j++)  
        a[i][j] = x[i][j];
```

```
factor(a,n,b,ipivot,iflag);
```

```
if (iflag == 2)
```

```
{printf("MATRIX IS SINGULAR\n");  
return;  
}
```

```
for (i=1; i<=n; i++)
```

```
{ b[i] = 0.;  
}
```

```
ibeg = 1;
```

```

for (j=1; j<=n; j++)
    {
        b[j] = 1.;
        subst(a,ipivot,b,n,ainv,j);
        b[j] = 0.;
        ibeg = ibeg + n;
    }
}

/* FUNCTION FACTOR SUPPORTS THE FUNCTION INVERT */
/* factor(a,n,b,ipivot,iflag); */
factor(float ww[35][35],int n, float d[35],int ipivot[35],
        int iflag)
{
    float rowmax, colmax, awikov, temp, ratio;
    int i, j, k, istar, nml, kpl, ip, ipk;

    /* INITIALIZE IPIVOT, D */
    for (i=1; i<=n; i++)
        {
            ipivot[i] = i;
            rowmax = 0.;

            for (j=1; j<=n; j++)
                {
                    if (rowmax < fabs(ww[i][j]))
                        rowmax = fabs(ww[i][j]);
                }

            if (rowmax == 0.0)
                {
                    iflag = 2;
                    return; /*goto done;*/
                }

            d[i] = rowmax;
        }

    /* GAUSS ELIMINATION WITH SCALED PARTIAL PIVOTING */

    if (n <= 1) return;

```



```

/* DETERMINE PIVOT ROW, THE ROW ISTAR */
for (k=1; k<=(n-1); k++)

    { colmax = fabs(ww[k][k]) / d[k];
      istar = k;

      for (i=(k+1); i<=n; i++)

          { awikov = fabs(ww[i][k]) / d[i];

            if (awikov > colmax)
              { colmax = awikov;
                istar = i;
              }
          }

      if (colmax == 0.0)

          { iflag = 2;
            return;
          }

      if (istar > k) /* MAKE K THE PIVOT ROW AND INTERCHANGE
*/

          { /* iflag = -iflag; */
            i = ipivot[istar];
            ipivot[istar] = ipivot[k];
            ipivot[k] = i;

            temp = d[istar];
            d[istar] = d[k];
            d[k] = temp;

            for (j=1; j<=n; j++)

                { temp = ww[istar][j];
                  ww[istar][j] = ww[k][j];
                  ww[k][j] = temp;
                }
          }

/* ELIMINATE X[K] FROM ROW K+1, ..., N */

      for (i=k+1; i<=n; i++)

          { ww[i][k] = ww[i][k] / ww[k][k];

```

```

        ratio = ww[i][k];
        for (j=(k+1); j<=n; j++)
            { ww[i][j] = ww[i][j] - ratio*ww[k][j];
              }
    }

if (ww[n][n] == 0.0) iflag = 2;

}

/* THE FUNCTION SUBST IS USED AS A SUBSTITUTION ALGORITHM
FOR INVERT */

/*  subst(a,ipivot,b,n,ainv[ibeg]); */

subst(float ww[35][35],int ipivot[35], float b[35], int n,
float x[35][35],
    int jj)

{
    int i, j, k;
    int ip, kml, npl, nplmk, kpl;
    float sum;

    if (n <= 1)
        { x[1][1] = b[1] / ww[1][1];
          return; /* goto done; */
        }

    ip = ipivot[1];
    x[1][jj] = b[ip];

    for (i=2; i<=n; i++)
        { sum = 0.;

          for (j=1; j<=(i-1); j++)
              sum = ww[i][j] * x[j][jj] + sum;

          ip = ipivot[i];

```

```

        x[i][jj] = b[ip] - sum;
    }
x[n][jj] = x[n][jj] / ww[n][n];

for (i=(n-1); i>=1; i--)
{
    sum = 0.;
    for (j=(i+1); j<=n; j++)
        sum = ww[i][j] * x[j][jj] + sum;
    x[i][jj] = (x[i][jj]-sum) / ww[i][i];
}

/* done:*/
}

```

REGOUT

The following seven files comprise the REGOUT executable module. Regout1 is the driver for the executable module and the Aptness Assessment menu. Regout1a controls the Model Results menu. Regout2 calculates the graphical information and provides the printing capability for the graphs. Regout3, Regout4, and Regout5 provide the functions that handle the regression output displays. Regout6 performs the decoding of the regression coefficients and the output displays of the Decoded Coefficient Table.

REGOUT1.C

```
/* REGOUT.EXE PRODUCES LINEAR REGRESSION OUTPUT */
/* REGOUT1.C CONTROLS LINEAR REGRESSION OUTPUT */
/* MODIFICATIONS:
    19 DEC 90 - aptness and model results broken out into
                separate file REGOUT1A.C to solve memory
problem
                during compile */

# include <stdio.h>
# include <math.h>
# include <dos.h>
# include <string.h>

/* MAIN CONTROLS THE OUTPUT PROGRAM REGOUT.EXE */

main ()
{
    FILE *infile, *outfile, *file;
    int i,j,k,l,m1,n1,n2,nn,p,q,r; /* COUNTERS */
    int m, n; /* # ROWS & COLUMNS */

    int totddf, regddf, errddf; /* DEGREES OF FREEDOM */
    int sstocdf, ssrddf, ssecdf;
```

```

float ssr, ssto, sse;
float ssrsrc, sstoc, ssec;
float msto, msr, mse;
float msrsrc, msec;
float freg, fregp, rsq, rsqa;
float sum, ybar, mm;

static float x[35][35];
static float y[35][2];
static float beta[35][2];          /* BETA ESTIMATES */
float betap[2][35];                /* BETAP */
float ss[2][2];                    /* SS scratch pad */
static float xpxinv[35][35];

static float stats[35][10];        /* REGRESSION STATISTICS */
static float ssrftest[35];
static float xtrassr[35][2];       /* EXTRA SS */
static float covcorr[35][35];      /* COVARIANCE/CORRELATION
MATRIX */

float sspe, mspe;                  /* LACK OF FIT PARAMETERS */
float sslf, mslf;
float flof, flofp;
int level, row, row1;
int facs, totrow;
int reps, creps;
float reps1, creps1;
int c, sspedf, sslfdf;

int gpscreen;
int factors;
int orthog;                        /* ORTHOGONAL DESIGN = 1 */
int choice;                        /* CHOOSE THE DESIRED
TRANSFORMATION */

char regmenu='1';
char ch;
char outname[24], output[10];

/* READ FROM FILE EXP.DES */

if ((infile = fopen("regress.out", "r")) == NULL)
{ printf("CAN'T OPEN REGRESSION FILE\n");
  return;
}

fscanf(infile, "%d\n", &m);
fscanf(infile, "%d\n", &n);
fscanf(infile, "%d\n", &facs);

```

```

fscanf(infile,"%d\n",&orthog);
fscanf(infile,"%d\n",&choice);
fscanf(infile,"%d\n",&level);
fscanf(infile,"%d\n",&creps);
fscanf(infile,"%d\n",&reps);
fscanf(infile,"%d\n",&c);
fscanf(infile,"%d\n",&row);

fscanf(infile,"%g\n",&ssrc);
fscanf(infile,"%d\n",&ssrcdf);
fscanf(infile,"%g\n",&msrc);
fscanf(infile,"%g\n",&freg);
fscanf(infile,"%g\n",&ssec);
fscanf(infile,"%d\n",&ssecdf);
fscanf(infile,"%g\n",&msec);
fscanf(infile,"%g\n",&sstoc);
fscanf(infile,"%d\n",&sstocdf);
fscanf(infile,"%g\n",&fregp);
fscanf(infile,"%g\n",&rsq);
fscanf(infile,"%g\n",&rsqa);

fscanf(infile,"%g\n",&sslf);
fscanf(infile,"%d\n",&sslfd);
fscanf(infile,"%g\n",&mself);
fscanf(infile,"%g\n",&sspe);
fscanf(infile,"%d\n",&sspedf);
fscanf(infile,"%g\n",&mspe);
fscanf(infile,"%g\n",&flop);
fscanf(infile,"%g\n",&flopfp);

for (i=1; i<=m; i++)
    for (j=1; j<=n; j++)
        fscanf(infile,"%g %c",&x[i][j],&ch);

fscanf(infile,"\n");

for (i=1; i<=m; i++)
    fscanf(infile,"%g %c",&y[i][1],&ch);

fscanf(infile,"\n");

for (i=1; i<=m; i++)
    for (j=0; j<=9; j++)
        fscanf(infile,"%g %c",&stats[i][j],&ch);

fscanf(infile,"\n");

for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)

```



```

printf("          \272          (3) EXIT
      \272\n");
printf("          \272"); for (k=2; k<=50; k++) printf(" ");
printf("\272\n");
printf("          \310"); for (k=2; k<=50; k++)
printf("\315");
printf("\274\n\n\n\n\n\n\n\n\n");

regmenu = getch();

switch(regmenu)
{
case '1':
    apt(stats,y,m,n,nn,msec,outfile);
break;

case '2':
    mod
    (stats,x,y,beta,xpxinv,xtrassr,ssrfest,m,n,facs,orthog,choi
ce,level,

creps,reps,c,row,ssrc,ssrcdf,msrc,freg,ssec,ssecdf,msec,ssto
c,sstocdf,

fregp,rsq,rsqa,sslf,sslfdf,mslf,sspe,sspedf,mspe,flof,flofp,
outname,
    outfile);

break;

case '3':
break;

default;;
}    /* switch */
}    /* while */

fclose(outfile);
}

```

REGOUT1A.C

/* REGOUT1A.C CONTROLS LINEAR REGRESSION OUTPUT WITH
REGOUT1.C */

/* MODIFICATAIONS:

19 DEC 90 - REGOUT1A.C created to solve memory problem
on
compile.


```

aptmenu = getch();

switch (aptmenu)
{
case '1':

/* CALCULATE AND PLOT
Y-HAT (stats[i][6]) vs STANDARDIZED RESIDUALS
(stats[i][8]) */

for (i=1; i<=m; i++)
{ stats[i][7] = y[i][1] - stats[i][6];
  stats[i][8] = stats[i][7]/sqrt(msec);
}

plot(stats,m,6,8,10.,outfile);

break;

case '2':

/* CALCULATE AND PLOT Y-HAT (stats[i][6]) vs RESIDUALS
(stats[i][7]) */

for (i=1; i<=m; i++)
  stats[i][7] = y[i][1] - stats[i][6];

plot(stats,m,6,7,10.,outfile);

break;

case '3':

/* CALCULATE AND PLOT RANKITS (stats[i][9]) vs
STANDARDIZED RESIDUALS (stats[i][8]) */

for (i=1; i<=m; i++)
{ stats[i][7] = y[i][1] - stats[i][6];
  stats[i][8] = stats[i][7]/sqrt(msec);
}

/* SORT THE STD RESIDUALS */

for (i=1; i<=m-1; i++)
{ k = i;
  for (nn=7; nn<=8; nn++)
    stats[0][nn] = stats[i][nn];
  for (j = i+1; j<=m; j++)
    { if (stats[j][8] < stats[0][8])

```

```

        { k = j;
          for (nn=7; nn<=8; nn++)
            stats[0][nn] = stats[j][nn];
        }
      }
    for (nn=7; nn<=8; nn++)
      { stats[k][nn] = stats[i][nn];
        stats[i][nn] = stats[0][nn];
      }
    }

/* CALCULATE THE WILK-SHAPIRO STATISTIC */
wilky = wilk(m, stats);

plot(stats,m,8,9,wilky,outfile);

break;

case '4':
default:
break;
    } /* switch */
  } /* while */
return;
} /* END APT */

mod
(stats,x,y,beta,xpxinv,xtrassr,ssrftest,m,n,facs,orthog,choi
ce,level,

creps,reps,c,row,ssrc,ssrcdf,msrc,freg,ssec,ssecdf,msec,ssto
c,sstocdf,

fregp,rsq,rsqa,sslf,sslfdf,mslf,sspe,sspedf,mspe,flof,flofp,
outname,
  outfile)

int m, n; /* # ROWS & COLUMNS */

int sstocdf, ssrcdf, ssecdf; /* DEGREES OF FREEDOM */

float ssrc, sstoc, ssec;
float msrc, msec;
float freg, fregp, rsq, rsqa;

float x[35][35];
float y[35][2];

```

```

float beta[35][2];           /* BETA ESTIMATES */
float xpxinv[35][35];

float stats[35][10];        /* REGRESSION STATISTICS */
float ssrftest[35];
float xtrassr[35][2];       /* EXTRA SS */

float sspe, mspe;           /* LACK OF FIT PARAMETERS */
float sslf, mslf;
float flof, flofp;

int level, row;
int facs;
int reps, creps;
int c, sspedf, sslfdf;

int orthog;                 /* ORTHOGONAL DESIGN = 1 */
int choice;                 /* CHOOSE THE DESIRED
TRANSFORMATION */

char outname [24];
FILE *outfile;
{
int i,j,k,l,m1,n1,n2,nn,p,q,r; /* COUNTERS */

int totdf, regdf, errdf; /* DEGREES OF FREEDOM */
float ssr, ssto, sse;
float msto, msr, mse;

float sum, ybar, mm;

float betap[2][35];         /* BETAP */
float ss[2][2];             /* SS scratch pad */

static float covcorr[35][35]; /* COVARIANCE/CORRELATION
MATRIX */
void decode();

int row1, totrow;
float reps1, creps1;

int gpscreen;
int factors;

char modmenu='1';
char ch;
char output[10];

```



```

printf("\n\311"); for (k=2; k<=65; k++) printf("\315");
printf("\273\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272                                ANALYSIS OF VARIANCE
                                \272\n");
if (m-n == 0)
{printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272    WARNING ! STATISTICAL TESTS ARE INVALID.
SSE = 0 WITH    \272\n"); }
if (m-n == 0)
printf("\272    df = 0.  THEY HAVE BEEN ARBITRARILY SET TO
ONE.    \272\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272    SOURCE                SS                df                MS
F    \272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272    Regression%12.3f        %3d        %12.3f        %8.2f
\272\n",

ssrc,ssrcdf,msrc,freg);
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
if (orthog == 0  && n > 10) n2 = n - 9;
else n2 = 2;
for (j=n; j>=2; j--)
{ printf("\272    X%-6.0f %12.3f        1        %12.3f
%8.2f    \272\n",

xtrassr[j][0],xtrassr[j][1],xtrassr[j][1],ssrfstest[j]);
if (j == n-15 || j == n-30)
{ printf("<RETURN>\r"); getch(); printf("
\r"); }
}
if ( (n >= 11 && n < 15) || (n2 >= 11 && n2 < 15) )
{ printf("<RETURN>\r"); getch(); printf("
\r"); }

printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272    Error                %12.3f        %3d        %12.3f
\272\n",ssec,ssecdf,msec);
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");

```

```

printf("\272 Total      %12.3f      %3d
      \272\n",sstoc,sstocdf);
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("<RETURN>\r"); getch(); printf("      \r");
if (orthog == 1)
{ printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
  printf("\272  ORTHOGONAL DESIGN
        \272\n"); }
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272 MODEL F VALUE =  %9.3f      P-VALUE = %7.4f
      \272\n",freg,fregp);
printf("\272  R SQUARED =      %6.3f
      \272\n",rsq);
printf("\272  ADJUSTED R SQUARED =%6.3f
      \272\n",rsqa);
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\310"); for (k=2; k<=65; k++) printf("\315");
printf("\274\n");

/* SEND TO FILE OR PRINTER */

printf("\n1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();
printf("\n");

if (ch == '1')
{
  fprintf(outfile,"\n\n\n\311"); for (k=2; k<=65; k++)
  fprintf(outfile,"\315"); fprintf(outfile,"\273\n");
  fprintf(outfile,"\272"); for (k=2; k<=65; k++)
  fprintf(outfile," "); fprintf(outfile,"\272\n");
  fprintf(outfile,"\272      ANALYSIS OF
VARIANCE      \272\n");
  if (m-n == 0)
  { fprintf(outfile,"\272"); for (k=2; k<=65; k++)
  fprintf(outfile," "); fprintf(outfile,"\272\n");
  fprintf(outfile,"\272  WARNING ! STATISTICAL TESTS ARE
INVALID.  SSE = 0 WITH      \272\n"); }
  if (m-n == 0)
  fprintf(outfile,"\272  df = 0.  THEY HAVE BEEN ARBITRARILY
SET TO ONE.      \272\n");
  fprintf(outfile,"\272"); for (k=2; k<=65; k++)
  fprintf(outfile," "); fprintf(outfile,"\272\n");

```

```

fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272 SOURCE SS df
MS F \272\n");
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272 Regression%12.3f %3d %12.3f
%8.2f \272\n",

ssrc, ssrddf, msr, freg);
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
if (orthog == 0 && n > 10) n2 = n - 9;
else n2 = 2;
for (j=n; j>=2; j--)
{ fprintf(outfile, "\272 X%-6.0f %12.3f 1
%12.3f %8.2f \272\n",

xtrassr[j][0], xtrassr[j][1], xtrassr[j][1], ssrftest[j]);
}
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272 Error %12.3f %3d %12.3f
\272\n", ssec, ssecdf, msec);
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272 Total %12.3f %3d
\272\n", sstoc, sstocdf);
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
if (orthog == 1)
{ fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\272 ORTHOGONAL DESIGN
\272\n"); }
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\272 MODEL F VALUE = %9.3f P-VALUE =
%7.4f \272\n", freg, fregp);
fprintf(outfile, "\272 R SQUARED = %6.3f
\272\n", rsq);
fprintf(outfile, "\272 ADJUSTED R SQUARED =%6.3f
\272\n", rsqa);
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\310"); for (k=2; k<=65; k++)
fprintf(outfile, "\315"); fprintf(outfile, "\274\n");
}
else if (ch == '2')
{

```



```

fprintf(stdprn, "\n\n\n\r");
for (k=2; k<=65; k++) fprintf(stdprn, "\315");
fprintf(stdprn, "\n\n\r          ANALYSIS OF
VARIANCE          \r\n");
if (m-n == 0)
fprintf(stdprn, "          WARNING ! STATISTICAL TESTS ARE INVALID.
SSE = 0 WITH \r\n");
if (m-n == 0)
fprintf(stdprn, "          df = 0.  THEY HAVE BEEN ARBITRARILY SET
TO ONE. \r\n");
fprintf(stdprn, "\r\n");
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n SOURCE          SS          df
MS          F          \r\n");
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n Regression%12.3f          %3d          %12.3f
%8.2f \r\n\r\n",

ssrc, ssrddf, msr, freg);
if (orthog == 0 && n > 10) n2 = n - 9;
else n2 = 2;
for (j=n; j>=2; j--)
{ fprintf(stdprn, "          X%-6.0f %12.3f          1          %12.3f
%8.2f \r\n",

xtrassr[j][0], xtrassr[j][1], xtrassr[j][1], ssrftest[j]);
}
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n Error          %12.3f          %3d          %12.3f
\r\n", ssec, ssecdf, msec);
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n Total          %12.3f          %3d
\r\n", sstoc, sstocdf);
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n");
if (orthog == 1)
fprintf(stdprn, "\r\n ORTHOGONAL DESIGN
\r\n");
fprintf(stdprn, "\r\n MODEL F VALUE = %9.3f          P-VALUE =
%7.4f          \r\n", freg, fregp);
fprintf(stdprn, "          R SQUARED =          %6.3f
\r\n", rsq);
fprintf(stdprn, "          ADJUSTED R SQUARED =%6.3f
\r\n", rsqa);
fprintf(stdprn, "\r\n");
for (k=2; k<=65; k++) fprintf(stdprn, "\315");
}

break;

```

```

case '2':

/* VARIANCES (stats[i][2]) and STANDARD ERRORS (stats[i][1])
  OF THE COEFFICIENTS */

printf("\n\311"); for (k=2; k<=65; k++) printf("\315");
printf("\273\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272                                COEFFICIENT TABLE
                                \272\n");
if (m-n == 0)
  {printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272      WARNING ! STATISTICAL TESTS ARE INVALID.
SSE = 0 WITH      \272\n"); }
if (m-n == 0)
printf("\272      df = 0.  THEY HAVE BEEN ARBITRARILY SET TO
ONE.      \272\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272 VARIABLE      VALUE      STD ERROR      VARIANCE
STUDENT-T  P-VALUE  \272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");

k=1;
for (i=1; i<=n; i++)
  { printf("\272 X%-6.0f%11.3f  %10.3f %10.3f  %7.3f
%7.4f  \272\n",

stats[i][5],stats[i][0],stats[i][1],stats[i][2],stats[i][3],
stats[i][4]);
    k = k+1;
    if (k == 15 || k == 32)
      { printf("<RETURN>\r"); getch();
printf("      \r"); }
  }
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\310"); for (k=2; k<=65; k++) printf("\315");
printf("\274\n");

/* SEND TO FILE OR PRINTER */

```

```

printf("\n1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();
printf("\n");

if (ch == '1')
{
fprintf(outfile,"\n\n\n\311"); for (k=2; k<=65; k++)
fprintf(outfile,"\315"); fprintf(outfile,"\273\n");
fprintf(outfile,"\272"); for (k=2; k<=65; k++)
fprintf(outfile," "); fprintf(outfile,"\272\n");
fprintf(outfile,"\272                                COEFFICIENT TABLE
                                \272\n");
if (m-n == 0)
{fprintf(outfile,"\272"); for (k=2; k<=65; k++)
fprintf(outfile," "); fprintf(outfile,"\272\n");
fprintf(outfile,"\272      WARNING ! STATISTICAL TESTS ARE
INVALID.  SSE = 0 WITH      \272\n"); }
if (m-n == 0)
fprintf(outfile,"\272      df = 0.  THEY HAVE BEEN ARBITRARILY
SET TO ONE.      \272\n");
fprintf(outfile,"\272"); for (k=2; k<=65; k++)
fprintf(outfile," "); fprintf(outfile,"\272\n");
fprintf(outfile,"\307"); for (k=2; k<=65; k++)
fprintf(outfile,"\304"); fprintf(outfile,"\266\n");
fprintf(outfile,"\272 VARIABLE      VALUE      STD ERROR
VARIANCE STUDENT-T P-VALUE \272\n");
fprintf(outfile,"\307"); for (k=2; k<=65; k++)
fprintf(outfile,"\304"); fprintf(outfile,"\266\n");
fprintf(outfile,"\272"); for (k=2; k<=65; k++)
fprintf(outfile," "); fprintf(outfile,"\272\n");

k=1;
for (i=1; i<=n; i++)
{ fprintf(outfile,"\272 X%-6.0f%11.3f  %10.3f %10.3f
%7.3f  %7.4f  \272\n",

stats[i][5],stats[i][0],stats[i][1],stats[i][2],stats[i][3],
stats[i][4]);
k = k+1;
}
fprintf(outfile,"\272"); for (k=2; k<=65; k++)
fprintf(outfile," "); fprintf(outfile,"\272\n");
fprintf(outfile,"\310"); for (k=2; k<=65; k++)
fprintf(outfile,"\315"); fprintf(outfile,"\274\n");

}
else if (ch == '2')
{

```



```

corrmat(msec,xpxinv,orthog,covcorr,n);

corr(beta,covcorr,n,outname,outfile);

break;

case '6':

if (creps > 0 || reps > 0)
{
/* ANALYSIS OF VARIANCE WITH LACK OF FIT */

printf("\n\311"); for (k=2; k<=65; k++) printf("\315");
printf("\273\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272          ANALYSIS OF VARIANCE WITH LACK OF FIT
\272\n");
if (m-n == 0)
{printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272          WARNING ! STATISTICAL TESTS ARE INVALID.
SSE = 0 WITH          \272\n"); }
if (m-n == 0)
printf("\272          df = 0.  THEY HAVE BEEN ARBITRARILY SET TO
ONE.          \272\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272  SOURCE          SS          df          MS
F          \272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272  Regression%12.3f          %3d          %12.3f          %8.2f
\272\n",

ssrc,ssrcdf,msrc,freg);
printf("\272  Error          %12.3f          %3d          %12.3f
\272\n",ssec,ssecdf,msec);
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272  Lack of Fit%9.3f          %3d          %12.3f
%8.2f \272\n",

sslf,sslfdf,mself,flof);

```

```

printf("\272      Pure Error %9.3f      %3d      %12.3f
      \272\n",

sspe,sspedf,mspe);
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272 Total      %12.3f      %3d
      \272\n",sstoc,sstocdf);
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
if (orthog == 1)
{ printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272 ORTHOGONAL DESIGN
      \272\n"); }
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272 MODEL F VALUE =      %9.3f      P-VALUE =
%7.4f
      \272\n",freg,fregp);
printf("\272 LACK OF FIT F VALUE = %9.3f      P-VALUE =
%7.4f
      \272\n",flof,flofp);
printf("\272 R SQUARED =      %6.3f
      \272\n",rsq);
printf("\272 ADJUSTED R SQUARED =%6.3f
      \272\n",rsqa);
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\310"); for (k=2; k<=65; k++) printf("\315");
printf("\274\n");

/* SEND TO FILE OR PRINTER */

printf("\n1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();
printf("\n");

if (ch == '1')
{
fprintf(outfile,"\n\n\n\311"); for (k=2; k<=65; k++)
fprintf(outfile,"\315"); fprintf(outfile,"\273\n");
fprintf(outfile,"\272"); for (k=2; k<=65; k++)
fprintf(outfile," "); fprintf(outfile,"\272\n");
fprintf(outfile,"\272 ANALYSIS OF VARIANCE WITH
LACK OF FIT      \272\n");
if (m-n == 0)
{ fprintf(outfile,"\272"); for (k=2; k<=65; k++)
fprintf(outfile," "); fprintf(outfile,"\272\n");

```

```

fprintf(outfile, "\272      WARNING ! STATISTICAL TESTS ARE
INVALID.  SSE = 0 WITH      \272\n"); }
if (m-n == 0)
fprintf(outfile, "\272      df = 0.  THEY HAVE BEEN ARBITRARILY
SET TO ONE.      \272\n");
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272  SOURCE      SS      df
      MS      F      \272\n");
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272  Regression%12.3f      %3d      %12.3f
      %8.2f      \272\n",

ssrc, ssrddf, msrc, freg);
fprintf(outfile, "\272  Error      %12.3f      %3d      %12.3f
      \272\n",

ssec, ssecdf, msec);
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272  Lack of Fit%9.3f      %3d      %12.3f
      %8.2f      \272\n",

sslf, sslfdf, mslf, flof);
fprintf(outfile, "\272  Pure Error %9.3f      %3d      %12.3f
      \272\n",

sspe, sspedf, mspe);
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272  Total      %12.3f      %3d
      \272\n", sstoc, sstocdf);
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
if (orthog == 1)
{ fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\272  ORTHOGONAL DESIGN
      \272\n"); }
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\272  MODEL F VALUE =      %9.3f
P-VALUE = %7.4f      \272\n", freg, fregp);
fprintf(outfile, "\272  LACK OF FIT F VALUE = %9.3f
P-VALUE = %7.4f      \272\n", flof, flofp);
fprintf(outfile, "\272  R SQUARED =      %6.3f
      \272\n", rsq);

```

```

fprintf(outfile, "\272 ADJUSTED R SQUARED =%6.3f
\272\n", rsqa);
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\310"); for (k=2; k<=65; k++)
fprintf(outfile, "\315"); fprintf(outfile, "\274\n");

}

else if (ch == '2')

{
fprintf(stdprn, "\n\n\n\r");
for (k=2; k<=65; k++) fprintf(stdprn, "\315");
fprintf(stdprn, "\n\n\r ANALYSIS OF VARIANCE WITH LACK OF
FIT \r\n");
if (m-n == 0)
fprintf(stdprn, " WARNING ! STATISTICAL TESTS ARE INVALID.
SSE = 0 WITH \r\n");
if (m-n == 0)
fprintf(stdprn, " df = 0. THEY HAVE BEEN ARBITRARILY SET
TO ONE. \r\n");
fprintf(stdprn, "\r\n");
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n SOURCE SS df
MS F \r\n");
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n Regression%12.3f %3d %12.3f
%8.2f ",
ssrc, ssrddf, msrc, freg);
fprintf(stdprn, "\r\n Error %12.3f %3d %12.3f
\r\n", ssec, ssecdf, msec);
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n Lack of Fit%9.3f %3d %12.3f
%8.2f ", sslf, sslfdf, mslf, flof);
fprintf(stdprn, "\r\n Pure Error %9.3f %3d
%12.3f\r\n", sspe, sspedf, mspe);
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n Total %12.3f %3d
\r\n", sstoc, sstocdf);
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\r\n");
if (orthog == 1)
fprintf(stdprn, "\r\n ORTHOGONAL DESIGN
\r\n");
fprintf(stdprn, "\r\n MODEL F VALUE = %9.3f P-VALUE
= %7.4f \r\n", freg, fregp);
fprintf(stdprn, " LACK OF FIT F VALUE =%9.3f P-VALUE =
%7.4f \r\n\n", flof, flofp);

```



```

fprintf(stdprn,"  R SQUARED =          %6.3f
          \r\n",rsq);
fprintf(stdprn,"  ADJUSTED R SQUARED =%6.3f
          \r\n",rsqa);
fprintf(stdprn,"\r\n");
for (k=2; k<=65; k++) fprintf(stdprn,"\315");

    }

}
else
{printf("LACK OF FIT UNAVAILABLE WITHOUT REPS <RETURN>\n");
  getch();
}

break;

case '7':

/* PRINT DESIGN MATRIX, Y, AND Y-HAT */

printf("\n\311"); for (k=2; k<=60; k++) printf("\315");
printf("\273\n");
printf("\272"); for (k=2; k<=60; k++) printf(" ");
printf("\272\n");
printf("\272                                DESIGN MATRIX
          \272\n");
if (level == 3)
{ printf("\272          NOTE: QUADRATICS TERMS HAVE BEEN
CORRECTED          \272\n");
printf("\272          SO THEY ARE ORTHOGONAL TO THE MEAN
          \272\n"); }
printf("\272"); for (k=2; k<=60; k++) printf(" ");
printf("\272\n");

n1 = 1;
if (n+2 > 8) n2 = 8;
else n2 = n+2;
while (n1 <= n+2)
{

printf("\307"); for (k=2; k<=60; k++) printf("\304");
printf("\266\n");
for (k=1; k<=60; k++) printf(" "); printf("\272\r");
printf("\272          ");
  for (i=n1; i<=n2; i++)
    { if (i <= n) printf("X%-5.0f",beta[i][0]);
      if (i == n+1) printf("          Y");
    }
}

```

```

        if (i == n+2) printf("          Y-HAT");
    }
    printf("\n\307"); for (k=2; k<=60; k++) printf("\304");
    printf("\266\n");

    for (i=1; i<=m; i++)
    { for (k=1; k<=60; k++) printf(" "); printf("\272\r");
      printf("\272  %3d",i);
      for (j=n1; j<=n2; j++)
      { if (j <= n) printf("%6.2f",x[i][j]);
        if (j == n+1) printf("%10.2f",y[i][1]);
        if (j == n+2) printf("%10.2f",stats[i][6]);
      }
      printf("\n");
      if (i == 17 || i == 34)
      { printf("<RETURN>\r"); getch(); printf("
\r"); }
    }
    printf("\272"); for (k=2; k<=60; k++) printf(" ");
    printf("\272\n");
    printf("\310"); for (k=2; k<=60; k++) printf("\315");
    printf("\274\n\n");
    printf("<RETURN>\r"); getch(); printf("          \r");
    n1 = n1+8;
    n2 = n2+8;
    if (n2 > n+2) n2 = n+2;
}

/* SEND TO FILE OR PRINTER */
printf("1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();
printf("\n");

if (ch == '1')
{
    fprintf(outfile,"\n\n\n\311"); for (k=2; k<=60; k++)
    fprintf(outfile,"\315"); fprintf(outfile,"\273\n");
    fprintf(outfile,"\272"); for (k=2; k<=60; k++)
    fprintf(outfile," "); fprintf(outfile,"\272\n");
    fprintf(outfile,"\272          DESIGN MATRIX
          \272\n");
    if (level == 3)
    { fprintf(outfile,"\272          NOTE: QUADRATICS TERMS HAVE
      BEEN CORRECTED          \272\n");
      fprintf(outfile,"\272          SO THEY ARE ORTHOGONAL TO THE
      MEAN          \272\n"); }
    fprintf(outfile,"\272"); for (k=2; k<=60; k++)
    fprintf(outfile," "); fprintf(outfile,"\272\n");
}

```

```

n1 = 1;
if (n+2 > 8) n2 = 8;
    else n2 = n+2;
while (n1 <= n+2)
{
    fprintf(outfile, "\307"); for (k=2; k<=60; k++)
    fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
    for (k=1; k<=60; k++) fprintf(outfile, " ");
    fprintf(outfile, "\272\r");
    fprintf(outfile, "\272      ");
        for (i=n1; i<=n2; i++)
        { if (i <= n)
    fprintf(outfile, "X%-5.0f", beta[i][0]);
            if (i == n+1) fprintf(outfile, "      Y");
            if (i == n+2) fprintf(outfile, "      Y-HAT");
        }
    fprintf(outfile, "\n\307"); for (k=2; k<=60; k++)
    fprintf(outfile, "\304"); fprintf(outfile, "\266\n");

        for (i=1; i<=m; i++)
        { for (k=1; k<=60; k++) fprintf(outfile, " ");
    fprintf(outfile, "\272\r");
        fprintf(outfile, "\272   %3d", i);
        for (j=n1; j<=n2; j++)
        { if (j <= n) fprintf(outfile, "%6.2f", x[i][j]);
            if (j == n+1) fprintf(outfile, "%10.2f", y[i][1]);
            if (j == n+2)
    fprintf(outfile, "%10.2f", stats[i][6]);
        }
        fprintf(outfile, "\n");
    }
    fprintf(outfile, "\272"); for (k=2; k<=60; k++)
    fprintf(outfile, " "); fprintf(outfile, "\272\n");
    fprintf(outfile, "\310"); for (k=2; k<=60; k++)
    fprintf(outfile, "\315"); fprintf(outfile, "\274\n\n\n");
    n1 = n1+8;
    n2 = n2+8;
    if (n2 > n+2) n2 = n+2;
}

}
else if (ch == '2')
{
    fprintf(stdprn, "\n\n\n\r");
    for (k=1; k<=60; k++) fprintf(stdprn, "\315");
    fprintf(stdprn, "\r\n\n\n      DESIGN MATRIX\n\r");
    if (level == 3)

```

```

{ fprintf(stdprn,"          NOTE: QUADRATICS TERMS HAVE BEEN
CORRECTED  \r\n");
fprintf(stdprn,"          SO THEY ARE ORTHOGONAL TO THE MEAN
\r\n"); }
fprintf(stdprn," \n");

n1 = 1;
if (n+2 > 8) n2 = 8;
    else n2 = n+2;
while (n1 <= n+2)
{

for (k=1; k<=60; k++) fprintf(stdprn," \304");
fprintf(stdprn," \n\r          ");
    for (i=n1; i<=n2; i++)
        { if (i <= n)
fprintf(stdprn,"X%-5.0f",beta[i][0]);
            if (i == n+1) fprintf(stdprn,"          Y");
            if (i == n+2) fprintf(stdprn,"          Y-HAT");
        }
fprintf(stdprn," \n\r");
for (k=1; k<=60; k++) fprintf(stdprn," \304");

    for (i=1; i<=m; i++)
        { fprintf(stdprn," \n\r   %3d",i);
            for (j=n1; j<=n2; j++)
                { if (j <= n) fprintf(stdprn,"%6.2f",x[i][j]);
                    if (j == n+1) fprintf(stdprn,"%10.2f",y[i][1]);
                    if (j == n+2)
fprintf(stdprn,"%10.2f",stats[i][6]);
                }
            }
        fprintf(stdprn," \n\r");
        for (k=1; k<=60; k++) fprintf(stdprn," \315");
        fprintf(stdprn," \r\n\n");
        n1 = n1+8;
        n2 = n2+8;
        if (n2 > n+2) n2 = n+2;
    }
}

break;

case '8':

/* PRINT THE INV(X'X) MATRIX */

xpxinvout(xpxinv, beta, n, outname, outfile);

break;

```

```
case '9':
```

```
default:
```

```
break;
```

```
    } /* SWITCH */
```

```
    } /* WHILE */
```

```
return;
```

```
} /* MOD */
```

REGOUT2.C

```
/* REGOUT2.C SUPPORTS LINEAR REGRESSION OUTPUT */
```

```
/* MODIFICATION HISTORY:
```

```
    8 JAN 91  -- PROVIDE ABILITY TO PRINT SCATTERPLOTS,  
ETC.
```

```
    BY          -- ROUILLARD    */
```

```
# include <stdio.h>
```

```
# include <math.h>
```

```
# include <dos.h>
```

```
# include <string.h>
```

```
/* FUNCTION MATMULmm MULTIPLIES MATRIX a (mxn) BY MATRIX b  
(nxm)
```

```
    AND CREATES MATRIX c (mxm) */
```

```
matmulmm(float a[35][35], float b[35][35], float c[35][35],  
          int m, int n, int p)
```

```
{  
    int i, j, k;  
    float sum;
```

```
    for (i=1; i<=m; i++)
```

```
        (for (j=1; j<=p; j++)
```

```
            {sum = 0.0;
```

```
                for (k=1; k<=n; k++)
```

```
                    { sum = sum + a[i][k] * b[k][j]; }
```

```
            c[i][j] = sum;
```

```
        }
```

```

    }
}

/* FUNCTION WILK CALCULATES THE WILK SHAPIRO STATISTIC FOR
   NORMALITY */

float wilk(n,res)
int n;
float res[35][10];
{
    float b[35], resbar, w, mpm, rtmpm, sum, wilky;
    int i;

    resbar = 0;
    sum = 0;
    mpm = 0;
    w = 0;

    for (i=1; i<=n; i++)
        resbar = resbar + res[i][8];

    resbar = resbar/n;

    for (i=1; i<=n; i++)
        sum = sum + (res[i][8] - resbar)*(res[i][8] - resbar);

    if (sum < .0001) sum = .001;

    for (i=1; i<=n; i++)
        mpm = mpm + res[i][9]*res[i][9];

    rtmpm = sqrt(mpm);

    for (i=1; i<=n; i++)
        b[i] = res[i][9]/rtmpm;

    for (i=1; i<=n; i++)
        w = w + b[i]*res[i][8];

    w = w*w;

    wilky = w/sum;

    return wilky;
}

```

```

/* SCATTERPLOT PROGRAM */

plot(stats,m,xcol,ycol,wilk,outfile)
float stats[35][10], wilk;
int m, xcol, ycol;
FILE *outfile;
{
char label[10][20], save;
/* char far *ptr = (char far *) 0xB8000000; pointer to CGA
memory */
int plot[35][10];
float x, y, min, max;
int minx, maxx, rangex, miny, maxy, rangey;
int i, j;
char ch, status();

strcpy(label[1],"STD ERROR");
strcpy(label[2],"VARIANCE");
strcpy(label[3],"T STATISTIC");
strcpy(label[4],"P-VALUE");
strcpy(label[5],"NAME");
strcpy(label[6],"Y-HAT");
strcpy(label[7],"RESID");
strcpy(label[8],"STD RES");
strcpy(label[9],"RANKITS");

/* find minimum, maximum, and range of the Xs */

min = stats[1][xcol];
for (i=2; i<=m; i++)
    if (stats[i][xcol] < min)    min = stats[i][xcol];
minx = floor(min);

max = stats[1][xcol];
for (i=2; i<=m; i++)
    if (stats[i][xcol] > max)    max = stats[i][xcol];
maxx = ceil(max);

rangex = maxx - minx;

if (rangex <= 1) rangex = 1;
else if (rangex > 10 && rangex <= 20) rangex = 20;
else if (rangex > 20 && rangex <= 50) rangex = 50;
else if (rangex > 50 && rangex <= 100) rangex = 100;

/* transform Xs into plotting coordinates */

if (minx == 0)

```

```

        {
            for (i=1; i<=m; i++)
                plot[i][xcol] = floor(stats[i][xcol]*200./rangex +
51);
        }
    else
    {
        for (i=1; i<=m; i++)
            plot[i][xcol] = floor( (stats[i][xcol] -
minx)*200./rangex + 50);
    }

/* find minimum, maximum, and range of the Ys */

min = stats[1][ycol];
for (i=2; i<=m; i++)
    if (stats[i][ycol] < min)    min = stats[i][ycol];
miny = floor(min);

max = stats[1][ycol];
for (i=2; i<=m; i++)
    if (stats[i][ycol] > max)    max = stats[i][ycol];
maxy = ceil(max);

rangey = maxy - miny;

if (rangey <= 1) rangey = 1;
else if (rangey > 8 && rangey <= 10) rangey = 10;
else if (rangey > 10 && rangey <= 20) rangey = 20;
else if (rangey > 20 && rangey <= 50) rangey = 50;
else if (rangey > 50 && rangey <= 100) rangey = 100;

/* transform Ys into plotting coordinates */

if (miny == 0)
{
    for (i=1; i<=m; i++)
        plot[i][ycol] = floor(169. -
stats[i][ycol]*150./rangey);
}
else
{
    for (i=1; i<=m; i++)
        plot[i][ycol] = floor(169. - (stats[i][ycol] -
miny)*150./rangey);
}

mode(4);
palette(1);

```



```

gridx(minx, rangex, label, xcol);
gridy(miny, rangey, wilk, label, xcol, ycol);

for (i=1; i<=m; i++)
{
    mempoint(plot[i][ycol]-1, plot[i][xcol], 2);
    mempoint(plot[i][ycol]+1, plot[i][xcol], 2);
    mempoint(plot[i][ycol], plot[i][xcol], 2);
    mempoint(plot[i][ycol], plot[i][xcol]-1, 2);
    mempoint(plot[i][ycol], plot[i][xcol]+1, 2);
}

save = getch();

mode(3);
printf("1) SEND TO PRINTER    2) EXIT");
printf("\n");
printf(" PRESS ANY KLY TO ABORT PRINTING ");

/* REDRAW AND PRINT SCREEN IF USER WANTS PLOT PRINTED */

ch = getch();
if (ch == '1')
{
    mode(4);
    palette(1);

    gridx(minx, rangex, label, xcol);
    gridy(miny, rangey, wilk, label, xcol, ycol);

    for (i=1; i<=m; i++)
    {
        mempoint(plot[i][ycol]-1, plot[i][xcol], 2);
        mempoint(plot[i][ycol]+1, plot[i][xcol], 2);
        mempoint(plot[i][ycol], plot[i][xcol], 2);
        mempoint(plot[i][ycol], plot[i][xcol]-1, 2);
        mempoint(plot[i][ycol], plot[i][xcol]+1, 2);
    }
    printscr();
}

mode(3);
}

gridx(minx, rangex, label, xcol)
int minx, rangex, xcol;
char label[][20];

{
    register int t, i, j;
    float x1, x2, x3, x4, x5, x6;

```

```

x1 = minx;
x2 = minx + .2*rangex;
x3 = minx + .4*rangex;
x4 = minx + .6*rangex;
x5 = minx + .8*rangex;
x6 = minx + rangex;

gotoxy(22,5); printf("%.1f", x1);
gotoxy(22,10); printf("%.1f", x2);
gotoxy(22,15); printf("%.1f", x3);
gotoxy(22,20); printf("%.1f", x4);
gotoxy(22,25); printf("%.1f", x5);
gotoxy(22,30); printf("%.1f",x6);
gotoxy(20,32); printf(label[xcol]);

line(170, 50, 170, 250, 1);

j = 50;
while (j<=250)
    { for (i=170; i<=173; i++)
        mempoint(i,j,1);
      j = j + 40;
    }
}

gridy(miny,rangey,wilk,label,xcol,ycol)
int miny, rangey, xcol, ycol;
float wilk;
char label[][20];
{
    register int t,i,j;
    float y1, y2, y3, y4, y5, y6;

    y1 = miny;
    y2 = miny + .2*rangey;
    y3 = miny + .4*rangey;
    y4 = miny + .6*rangey;
    y5 = miny + .8*rangey;
    y6 = miny + rangey;

    gotoxy( 1,5); printf(label[ycol]);
    gotoxy( 2,17); printf(label[xcol]);
                    printf(" vs "); printf(label[ycol]);
    gotoxy( 2,0); printf("%.1f", y6);
    gotoxy( 6,0); printf("%.1f", y5);
    gotoxy(10,0); printf("%.1f", y4);

```

```

gotoxy(14,0); printf("%.1f", y3);
gotoxy(17,0); printf("%.1f", y2);
gotoxy(21,0); printf("%.1f", y1);
if (wilk < 2.0) { gotoxy(24,0); printf("WILK SHAPIRO =
%.3f",wilk); }
/* gotoxy(24,22); printf("1)TO FILE 2)EXIT"); */

line(20, 50, 170, 50, 1);

i = 20;
while (i<=170)
    { for (j=48; j<=50; j++)
        mempoint(i,j,1);
      i = i + 30;
    }

}

line(startx, starty, endx, endy, color)
int startx, starty, endx, endy, color;
{
register int t, distance;
int x=0, y=0, deltax, deltay;
int incx, incy;

/* compute the distances in both directions */
deltax = endx - startx;
deltay = endy - starty;

/* compute direction (0 means vert or hor) */
if (deltax > 0) incx = 1;
else if (deltax == 0) incx = 0;
else incx = -1;

if (deltay > 0) incy = 1;
else if (deltay == 0) incy = 0;
else incy = -1;

/* determine which distance is greater */
deltax = abs(deltax);
deltay = abs(deltay);
if (deltax > deltay) distance = deltax;
else distance = deltay;

/* draw the line */
for (t=0; t<=distance+1; t++)
    { mempoint(startx, starty, color);
      x+= deltax;
      y+= deltay;
      if (x > distance)

```

```

        { x--distance;
          startx+=incx;
        }
    if (y > distance)
    { y--distance;
      starty+=incy;
    }
}
}

/* write point to screen */
mempoint(x, y, colorcode)
int x, y, colorcode;
{
union mask {
    char c[2];
    int i; } bitmask;
int i, index, bitposition;
unsigned char t;
char xor;
char far *ptr = (char far *) 0xB8000000;

bitmask.i = 0xFF3F;

/* check range for mode 4 */
if (x < 0 || x > 199 || y < 0 || y > 319) return;

xor = colorcode & 128;
colorcode = colorcode & 127;

/* set bitmask and colocode bits to right location */
bitposition = y%4;
colorcode <=> 2*(3-bitposition);
bitmask.i >=> 2*bitposition;

/* find the correct byte in screen memory */
index = x*40 +(y >> 2);
if (x % 2) index +=8152;

/* write the color */
if (!xor)
{ t = *(ptr+index) & bitmask.c[0];
  *(ptr+index) = t | colorcode;
}
else
{ t = *(ptr+index) | (char)0;
  *(ptr+index) = t ^ colorcode;
}
}

```

```

/* set video mode */
mode(modecode)
int modecode;
{
union REGS r;
r.h.al = modecode;
r.h.ah = 0;
int86(0x10, &r, &r);
}

/* send cursor to x,y */
gotoxy(x, y)
int x, y;
{

union REGS r;
r.h.ah = 2;
r.h.dl = y;
r.h.dh = x;
r.h.bh = 0;
int86(0x10, &r, &r);
}

/* set the palette */
palette(pnum)
int pnum;
{

union REGS r;
r.h.bh = 1;
r.h.bl = pnum;
r.h.ah = 11;
int86(0x10, &r, &r);
}

/* DETERMINE STATUS OF PRINTER */
char status()
{
    union REGS reg;

    reg.h.ah = 2;
    reg.x.dx = 0;
    int86(0x17, &reg, &reg);
    return(reg.h.ah & 0x80);
}

/* SEND CHAR TO PRINTER */
char put_out(char character)
{
    union REGS reg;

```

```

        while (!status());
        reg.h.ah = 0;
        reg.h.al = character;
        reg.x.dx = 0;
        int86(0x17, &reg,&reg);
        return (reg.h.ah);
    }

/* READ A PIXEL FROM THE SCREEN */
int readPixel(int x, int y)
{
    union REGS reg;
    reg.h.ah = 0x0D;
    reg.x.cx = x;
    reg.x.dx = y;
    int86 (0x10, &reg,&reg);
    return (reg.h.al);
}

/* PRINT SCREEN DISPLAY ON EPSON PRINTER */

int XMIN = 0, XMAX = 319, YMIN = 0, YMAX = 199;
unsigned char background;

printscr()
{
    int i,x,y;

    put_out(0x1b);
    put_out(0x33);
    put_out(0x15);
    for (i=0; i< 12; i++)
        put_out(0x0A);
    background = readPixel(0,0);
    for (x = XMAX; x > XMIN+3; x-=8)
    {
        if (kbhit())
        {
            put_out(0x01b);
            put_out(0x32);
            getch();
            break;
        }
        printrow(x);
    }
    put_out('\x0C');
}

```

```

printrow(int x)
{
    unsigned char savechar, temp;
    static unsigned char out_buff[434]={"\x1bK\xAE\x01"};
    unsigned int i, j, newy, y;

    for (y=YMIN,j=84; y<=YMAX; y++,j++)
    {
        savechar = 0;
        for (i=0; i<8; i++)
        {
            temp = readPixel(x-i,y);
            if (temp != background)
                savechar |= 1;
            if (i!=7)
                savechar <=< 1;
        }
        out_buff[j] = savechar;
    }
    for (i=0; i<434; i++)
        put_out(out_buff[i]);
    put_out('\r');
    put_out('\n');
}

```

REGOUT3.C

```

/* REGOUT3.C SUPPORTS LINEAR REGRESSION OUTPUT */

# include <stdio.h>
# include <math.h>
# include <dos.h>
# include <string.h>

/* FUNTION CORR CALCULATES THE CORRELATION MATRIX */
corr(beta,scratchy,n,outname,outfile)
float beta[35][2], scratchy[35][35];
int n;
char outname[24];
FILE *outfile;
{
    int i, j, k, n1, n2;
    char ch;

    printf("\n\311"); for (k=2; k<=67; k++) printf("\315");
    printf("\273\n");
}

```

```

printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");
printf("\272                                CORRELATION MATRIX
                                \272\n");
printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");

n1 = 1;
if (n > 8) n2 = 8;
    else n2 = n;
while (n1 <= n)
{

printf("\307"); for (k=2; k<=67; k++) printf("\304");
printf("\266\n");
for (k=1; k<=67; k++) printf(" "); printf("\272\r");
printf("\272                                ");
for (i=n1; i<=n2; i++) printf(" X%-5.0f",beta[i][0]);
printf("\n\307"); for (k=2; k<=67; k++) printf("\304");
printf("\266\n");

    for (i=1; i<=n; i++)
        { for (k=1; k<=67; k++) printf(" "); printf("\272\r");
          printf("\272 X%-5.0f",beta[i][0]);
          for (j=n1; j<=n2; j++)
              {printf("%7.3f",scratchy[i][j]);}
          printf("\n");
          if (i == 17 || i == 34)
              { printf("<RETURN>\r"); getch(); printf("
\r"); }
        }
    printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");
    printf("\310"); for (k=2; k<=67; k++) printf("\315");
printf("\274\n\n");
    printf("<RETURN>\r"); getch(); printf("                                \r");
    n1 = n1+8;
    n2 = n2+8;
    if (n2 > n) n2 = n;
}

/* SEND TO FILE OR PRINTER */
printf("1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();
printf("\n");

if (ch == '1')
{

```



```

fprintf(outfile, "\n\n\n\311"); for (k=2; k<=67; k++)
fprintf(outfile, "\315"); fprintf(outfile, "\273\n");
fprintf(outfile, "\272"); for (k=2; k<=67; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\272                                     CORRELATION
MATRIX                                     \272\n");
fprintf(outfile, "\272"); for (k=2; k<=67; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");

n1 = 1;
if (n > 8) n2 = 8;
    else n2 = n;
while (n1 <= n)
{

fprintf(outfile, "\307"); for (k=2; k<=67; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
for (k=1; k<=67; k++) fprintf(outfile, " ");
fprintf(outfile, "\272\r");
fprintf(outfile, "\272                                     ");
for (i=n1; i<=n2; i++) fprintf(outfile, "
X%-5.0f", beta[i][0]);
fprintf(outfile, "\n\307"); for (k=2; k<=67; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");

    for (i=1; i<=n; i++)
    { for (k=1; k<=67; k++) fprintf(outfile, " ");
fprintf(outfile, "\272\r");
    fprintf(outfile, "\272 X%-5.0f", beta[i][0]);
    for (j=n1; j<=n2; j++)
        { fprintf(outfile, "%7.3f", scratchy[i][j]); }
    fprintf(outfile, "\n");
    }
    fprintf(outfile, "\272"); for (k=2; k<=67; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
    fprintf(outfile, "\310"); for (k=2; k<=67; k++)
fprintf(outfile, "\315"); fprintf(outfile, "\274\n\n\n\n");
    n1 = n1+8;
    n2 = n2+8;
    if (n2 > n) n2 = n;
    }

}
else if (ch == '2')
{

    corrprrt(scratchy, beta, n);

}

```



```

int orthog,n;
{

float scratch[35][35];
float scratch2[35][35], scratch4[35][35];
int i, j, k, n1, n2;
char ch;

for (i=1; i<=n; i++)
    for (j=1; j<=n; j++)
        scratch[i][j] = msec * xpxinv[i][j];

if (orthog == 1)
{
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            scratchy[i][j] = 0;

    for (i=1; i<=n; i++)
        scratchy[i][i] = 1.;
}
else
{
    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            scratch4[i][j] = 0;

    for (i=1; i<=n; i++)
        scratch4[i][i] = 1./sqrt(scratch[i][i]);

    matmulmm(scratch4,scratch,scratch2,n,n,n);

    matmulmm(scratch2,scratch4,scratchy,n,n,n);
}
}

```

REGOUT4.C

```

/* REGOUT4.C SUPPORTS LINEAR REGRESSION OUTPUT */

# include <stdio.h>
# include <math.h>
# include <dos.h>
# include <string.h>

```

```

/* FUNTION COVAR CALCULATES THE VAR-COVARIANCE MATRIX */
covar(beta,scratch,n,outname,outfile)
float beta[35][2], scratch[35][35];
int n;
char outname[24];
FILE *outfile;
{

int i, j, k, n1, n2;
char ch;

printf("\n\311"); for (k=2; k<=67; k++) printf("\315");
printf("\273\n");
printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");
printf("\272                                VARIANCE-COVARIANCE MATRIX
                                \272\n");
printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");

n1 = 1;
if (n > 7) n2 = 7;
    else n2 = n;
while (n1 <= n)
{

printf("\307"); for (k=2; k<=67; k++) printf("\304");
printf("\266\n");
for (k=1; k<=67; k++) printf(" "); printf("\272\r");
printf("\272                                ");
for (i=n1; i<=n2; i++) printf(" X%-6.0f",beta[i][0]);
printf("\n\307"); for (k=2; k<=67; k++) printf("\304");
printf("\266\n");

    for (i=1; i<=n; i++)
    { for (k=1; k<=67; k++) printf(" "); printf("\272\r");
      printf("\272 X%-5.0f",beta[i][0]);
      for (j=n1; j<=n2; j++)
          (printf("%8.3f",scratch[i][j]));
      printf("\n");
      if (i == 17 || i == 34)
          ( printf("<RETURN>\r"); getch(); printf("
\r"); )
    }
    printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");
    printf("\310"); for (k=2; k<=67; k++) printf("\315");
printf("\274\n\n");
    printf("<RETURN>\r"); getch(); printf("                                \r");
}
}

```

```

    n1 = n1+7;
    n2 = n2+7;
    if (n2 > n) n2 = n;
}

/* SEND TO FILE OR PRINTER */
printf("1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();
printf("\n");

if (ch == '1')
{
    fprintf(outfile,"\n\n\n\311"); for (k=2; k<=67; k++)
    fprintf(outfile,"\315"); fprintf(outfile,"\273\n");
    fprintf(outfile,"\272"); for (k=2; k<=67; k++)
    fprintf(outfile," "); fprintf(outfile,"\272\n");
    fprintf(outfile,"\272                                VARIANCE-COVARIANCE
MATRIX                                \272\n");
    fprintf(outfile,"\272"); for (k=2; k<=67; k++)
    fprintf(outfile," "); fprintf(outfile,"\272\n");

    n1 = 1;
    if (n > 7) n2 = 7;
    else n2 = n;
    while (n1 <= n)
    {
        fprintf(outfile,"\307"); for (k=2; k<=67; k++)
        fprintf(outfile,"\304"); fprintf(outfile,"\266\n");
        for (k=1; k<=67; k++) fprintf(outfile," ");
        fprintf(outfile,"\272\n");
        fprintf(outfile,"\272                                ");
        for (i=n1; i<=n2; i++) fprintf(outfile,"
X%-6.0f",beta[i][0]);
        fprintf(outfile,"\n\307"); for (k=2; k<=67; k++)
        fprintf(outfile,"\304"); fprintf(outfile,"\266\n");

        for (i=1; i<=n; i++)
        { for (k=1; k<=67; k++) fprintf(outfile," ");
        fprintf(outfile,"\272\n");
        fprintf(outfile,"\272 X%-5.0f",beta[i][0]);
        for (j=n1; j<=n2; j++)
        {fprintf(outfile,"%8.3f",scratch[i][j]);}
        fprintf(outfile,"\n");
        }
        fprintf(outfile,"\272"); for (k=2; k<=67; k++)
        fprintf(outfile," "); fprintf(outfile,"\272\n");
    }
}

```

```

        fprintf(outfile, "\310"); for (k=2; k<=67; k++)
fprintf(outfile, "\315"); fprintf(outfile, "\274\n\n\n");
        n1 = n1+7;
        n2 = n2+7;
        if (n2 > n) n2 = n;
    }

}
else if (ch == '2')
{
    covarprt(scratch, beta, n);
}

}

/* SEND COVARIANCE MATRIX TO PRINTER */

covarprt(scratch, beta, n)
float scratch[35][35], beta[35][2];
int n;
{
    int i, j, k, n1, n2;
    char ch;

    fprintf(stdprn, "\n\n\n\r");
    for (k=2; k<=67; k++) fprintf(stdprn, "\315");
    fprintf(stdprn, "\r\n\n
VARIANCE-COVARIANCE MATRIX          \n\r");

    n1 = 1;
    if (n > 7) n2 = 7;
    else n2 = n;
    while (n1 <= n)
    {

        fprintf(stdprn, "\r\n");
        for (k=2; k<=67; k++) fprintf(stdprn, "\304");
        fprintf(stdprn, "\n\r          ");
        for (i=n1; i<=n2; i++)
            fprintf(stdprn, " X%-6.0f", beta[i][0]);
        fprintf(stdprn, "\r\n");
        for (k=2; k<=67; k++) fprintf(stdprn, "\304");

        for (i=1; i<=n; i++)
        { fprintf(stdprn, "\r\n X%-5.0f", beta[i][0]);
          for (j=n1; j<=n2; j++)
              (fprintf(stdprn, "%8.3f", scratch[i][j]));
        }
    }
}

```

```

        fprintf(stdprn, "\r\n\n");
        for (k=2; k<=67; k++) fprintf(stdprn, "\315");
        fprintf(stdprn, "\r\n\n\n");
        n1 = n1+7;
        n2 = n2+7;
        if (n2 > n) n2 = n;
    }
}

```

/* COVMAT CALCULATES THE VAR-COVARIANCE MATRIX */

```

covmat(msec, xpxinv, scratch, n)
float msec, xpxinv[35][35], scratch[35][35];
int n;
{
    int i, j;

    for (i=1; i<=n; i++)
        for (j=1; j<=n; j++)
            scratch[i][j] = msec * xpxinv[i][j];
}

```

REGOUT5.C

/* REGOUT5.C SUPPORTS LINEAR REGRESSION OUTPUT */

```

# include <stdio.h>
# include <math.h>
# include <dos.h>
# include <string.h>

```

/* XPXINVOUT OUTPUTS THE INV(X'X) MATRIX */

```

xpxinvout(xpxinv, beta, n, outname, outfile)
float xpxinv[35][35], beta[35][2];
int n;
char outname[24];
FILE *outfile;
{
    int i, j, k, n1, n2;
    char ch;

    printf("\n\311"); for (k=2; k<=67; k++) printf("\315");
    printf("\273\n");
    printf("\272"); for (k=2; k<=67; k++) printf(" ");
    printf("\272\n");
}

```

```

printf("\272                                INV(X'X) MATRIX
                                \272\n");
printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");

n1 = 1;
if (n > 8) n2 = 8;
    else n2 = n;
while (n1 <= n)
{

printf("\307"); for (k=2; k<=67; k++) printf("\304");
printf("\266\n");
for (k=1; k<=67; k++) printf(" "); printf("\272\r");
printf("\272                                ");
for (i=n1; i<=n2; i++) printf(" X%-5.0f",beta[i][0]);
printf("\n\307"); for (k=2; k<=67; k++) printf("\304");
printf("\266\n");

    for (i=1; i<=n; i++)
    { for (k=1; k<=67; k++) printf(" "); printf("\272\r");
      printf("\272 X%-5.0f",beta[i][0]);
      for (j=n1; j<=n2; j++)
          {printf("%7.3f",xpxinv[i][j]);}
      printf("\n");
      if (i == 17 || i == 34)
          { printf("<RETURN>\r"); getch(); printf("
\r"); }
    }
    printf("\272"); for (k=2; k<=67; k++) printf(" ");
printf("\272\n");
printf("\310"); for (k=2; k<=67; k++) printf("\315");
printf("\274\n\n");
printf("<RETURN>\r"); getch(); printf("                                \r");
n1 = n1+8;
n2 = n2+8;
if (n2 > n) n2 = n;
}

/* SEND TO FILE OR PRINTER */

printf("1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();
printf("\n");

if (ch == '1')
{

```



```

fprintf(outfile,"\\n\\311"); for (k=2; k<=67; k++)
fprintf(outfile,"\\315"); fprintf(outfile,"\\273\\n");
fprintf(outfile,"\\272"); for (k=2; k<=67; k++)
fprintf(outfile," "); fprintf(outfile,"\\272\\n");
fprintf(outfile,"\\272"                                INV(X'X) MATRIX
                                \\272\\n");
fprintf(outfile,"\\272"); for (k=2; k<=67; k++)
fprintf(outfile," "); fprintf(outfile,"\\272\\n");

n1 = 1;
if (n > 8) n2 = 8;
    else n2 = n;
while (n1 <= n)
{

fprintf(outfile,"\\307"); for (k=2; k<=67; k++)
fprintf(outfile,"\\304"); fprintf(outfile,"\\266\\n");
for (k=1; k<=67; k++) fprintf(outfile," ");
fprintf(outfile,"\\272\\r");
fprintf(outfile,"\\272                                ");
for (i=n1; i<=n2; i++) fprintf(outfile,"
X%-5.0f",beta[i][0]);
fprintf(outfile,"\\n\\307"); for (k=2; k<=67; k++)
fprintf(outfile,"\\304"); fprintf(outfile,"\\266\\n");

    for (i=1; i<=n; i++)
    { for (k=1; k<=67; k++) fprintf(outfile," ");
fprintf(outfile,"\\272\\r");
    fprintf(outfile,"\\272 X%-5.0f",beta[i][0]);
    for (j=n1; j<=n2; j++)
        {fprintf(outfile,"%7.3f",xpxinv[i][j]);}
    fprintf(outfile,"\\n");
    }
    fprintf(outfile,"\\272"); for (k=2; k<=67; k++)
fprintf(outfile," "); fprintf(outfile,"\\272\\n");
    fprintf(outfile,"\\310"); for (k=2; k<=67; k++)
fprintf(outfile,"\\315"); fprintf(outfile,"\\274\\n\\n\\n");
    n1 = n1+8;
    n2 = n2+8;
    if (n2 > n) n2 = n;
    }

}
else if (ch == '2')
{

    xpxinvprt(xpxinv, beta, n);

}

```



```

# include <dos.h>
# include <string.h>
int nn;

/* FUNCTION TO DECODE BETA COEFFICIENTS INTO ORIGINAL
FACTOR SETTINGS */

decode(stats,beta,n,outfile,outname)
float stats [35][10];
float beta [35] [2];      /* BETA ESTIMATES */
                           /* COL 0 CONTAINS NAME OF VARIABLE */
                           /* COL 1 CONTAINS THE ESTIMATES */

int n;
char outname[24];
FILE *outfile;
{
    FILE *infile;
    char inname[24];
    float range[35][2];      /* LOW and HI factor settings
*/
    float s[35], midpt[35];  /* .5 RANGES and MIDPOINTS
*/
    float uncoded[35] [2];   /* NAMES and DECODED
ESTIMATES */
    int i, j, k, index, test, numvar, b0;
    char ch;

    printf("\nPLEASE PROVIDE NAME OF FILE CONTAINING FACTOR
SETTINGS:  ");
    scanf("%s",inname);
    printf("\n");

    if ((infile = fopen(inname,"r")) == NULL)
    { printf("CAN'T OPEN FACTOR SETTINGS FILE
<RETURN>\n");
      return;
    }

    fscanf(infile,"%i\n",&numvar);      /* READ NUMBER OF
FACTORS */
    for (i=1; i<=numvar; i++)
    { fscanf(infile,"%g %c %g
%c",&range[i][0],&ch,&range[i][1],&ch);
      fscanf(infile,"\n");
    }
    fclose(infile);

    if (numvar < 10)
    {

```

```

/* CALCULATE MIDPOINT AND .5 RANGE FOR ALL FACTORS */
for (i=1; i<=numvar;i++)
    { s[i] = .5 * (range[i][1] - range[i][0]);
      midpt[i] = s[i] + range[i][0];
    }

nn = n;
b0 = 0;

/* LOCATE INTERCEPT TERM */
for (i=1; i<=n; i++)
    { if (beta[i][0] == 0)
      { b0 = i;
        uncoded[b0][0] = 0;
        uncoded[b0][1] = beta[b0][1];
      }
    }
if (b0 == 0)          /* intercept variable selected out by
user */
    { b0 = ++nn;
      uncoded[b0][0] = 0;
      uncoded[b0][1] = 0;
    }

/* CALCULATE DECODED VALUES FOR LINEAR TERMS */
for (i=1; i<=n; i++)
    { if ((beta[i][0] <= numvar) && (i != b0))
      { uncoded[i][0] = beta[i][0];

        /* to insure appropriate S value and midpt
beta[i][0] used
as index to S and MIDPT arrays */

        uncoded[i][1] = beta[i][1] / s[beta[i][0]];
        uncoded[b0][1] = uncoded[b0][1] - (beta[i][1] *
midpt[beta[i][0]])/s[beta[i][0]];
      }
    }

/* DETERMINE INTERACTION TERMS */
for (index = 1; index <= n; index++)
    { if (beta[index][0] > numvar)
      { test = beta[index][0]/10;
        i = test;
        j = beta[index][0] - (test * 10);
        if (i <= numvar)
            twoway(beta,uncoded,s,midpt,b0,index,i,j);
        else
            { k = j;
              test = i/10;
            }
      }
    }

```

```

        j = i - (test * 10);
        i = test;
        if (i <= numvar)
            threeway(beta,uncoded,s,midpt,b0,index,i,j,k);
    }
}

printf("\n\311"); for (k=2; k<=65; k++) printf("\315");
printf("\273\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272                                DECODED COEFFICIENTS
                                \272\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\272      WARNING ! ADDITIONAL TERMS MAY HAVE BEEN
INTRODUCED      \272\n");
printf("\272                                DURING DECODING
                                \272\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272 VARIABLE      VALUE
                                \272\n");
printf("\307"); for (k=2; k<=65; k++) printf("\304");
printf("\266\n");
printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");

k=1;
for (i=1; i<=nn; i++)
{ printf("\272 Z%-6.0f%11.5f
                                \272\n",
        uncoded[i][0],uncoded[i][1]);
    k = k+1;
    if (k == 15 || k == 32)
        {printf("<RETURN>\r"); getch(); printf("                \r");
        }
}

printf("\272"); for (k=2; k<=65; k++) printf(" ");
printf("\272\n");
printf("\310"); for (k=2; k<=65; k++) printf("\315");
printf("\274\n");

/* SEND TO FILE OR PRINTER */

printf("\n1) SEND TO FILE %s      2) SEND TO PRINTER      3)
EXIT",outname);
ch = getch();

```

```

printf("\n");

if (ch == '1')
{
fprintf(outfile, "\n\n\n\311"); for (k=2; k<=65; k++)
fprintf(outfile, "\315"); fprintf(outfile, "\273\n");
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\272                                DECODED
COEFFICIENTS                                \272\n");
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\272                                WARNING ! ADDITIONAL TERMS MAY HAVE
BEEN INTRODUCED                                \272\n");
fprintf(outfile, "\272                                DURING DECODING
                                \272\n");
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272 VARIABLE                                VALUE
                                \272\n");
fprintf(outfile, "\307"); for (k=2; k<=65; k++)
fprintf(outfile, "\304"); fprintf(outfile, "\266\n");
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
k=1;
for (i=1; i<=nn; i++)
{ fprintf(outfile, "\272 Z%-6.0f%11.5f
                                \272\n",
                                uncoded[i][0], uncoded[i][1]);
  k = k+1;
}
fprintf(outfile, "\272"); for (k=2; k<=65; k++)
fprintf(outfile, " "); fprintf(outfile, "\272\n");
fprintf(outfile, "\310"); for (k=2; k<=65; k++)
fprintf(outfile, "\315"); fprintf(outfile, "\274\n");
}

else if (ch == '2')
{
fprintf(stdprn, "\n\n\n\r");
for (k=2; k<=65; k++) fprintf(stdprn, "\315");
fprintf(stdprn, "\r\n\n                                DECODED COEFFICIENTS
                                \n\r");
fprintf(stdprn, "\n                                WARNING ! ADDITIONAL TERMS MAY HAVE
BEEN INTRODUCED                                \n\r");
fprintf(stdprn, "                                DURING DECODING
                                \n\n\r");
}

```

```

for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\n\r VARIABLE      VALUE
                \n\r");
for (k=2; k<=65; k++) fprintf(stdprn, "\304");
fprintf(stdprn, "\n\n\r");

k=1;
for (i=1; i<=nn; i++)
    { fprintf(stdprn, " Z%-6.0f%11.5f
        \n\r",
        uncoded[i][0], uncoded[i][1]
        );
      k = k+1;
    }
fprintf(stdprn, "\n\r");
for (k=2; k<=65; k++) fprintf(stdprn, "\315");

}
}
else
    { printf(" MORE THAN 9 VARIABLES <RETURN>\n");
      ch = getch();
    }
} /* END FUNCTION DECODE */

/* FUNCTION TWOWAY TO CALCULATE TWO-WAY INTERACTION AND
LOWER ORDERED TERMS */

twoway(beta, uncoded, s, midpt, b0, index, i, j)

float beta[35][2], uncoded[35][2], s[35], midpt[35];
int b0, index, i, j;
{
    int loc_i, loc_j, count;

    loc_i = 0;
    loc_j = 0;

    /* UPDATE INTERCEPT */
    uncoded[b0][1] = uncoded[b0][1] +
(beta[index][1]*midpt[i]*midpt[j])/(s[i]*s[j]);

    /* UNDATE INTERACTION TERM */
    uncoded[index][0] = (i * 10) + j;
    uncoded[index][1] = beta[index][1]/(s[i]*s[j]);

    /* LOCATE OR CREATE LINEAR TERMS */
    for (count = 1; count <= nn; count++)
        { if (uncoded[count][0] == i)

```

```

        { loc_i = count;
          uncoded[loc_i][1] = uncoded[loc_i][1] -
(beta[index][1]*midpt[j])/(s[i]*s[j]);
        }
        if (uncoded[count][0] == j)
        { loc_j = count;
          uncoded[loc_j][1] = uncoded[loc_j][1] -
(beta[index][1]*midpt[i])/(s[i]*s[j]);
        }
      }
      if (loc_i == 0)
      { nn++;
        uncoded[nn][0] = i;
        uncoded[nn][1] =
-(beta[index][1]*midpt[j])/(s[i]*s[j]);
      }
      if (loc_j == 0)
      { nn++;
        uncoded[nn][0] = j;
        uncoded[nn][1] =
-(beta[index][1]*midpt[i])/(s[i]*s[j]);
      }
    }
  } /* END FUNCTION TWOWAY */

/* FUNCTION THREEWAY TO CALCULATE THREE-WAY INTERACTIONS AND
LOWER ORDERS */

threeway(beta,uncoded,s,midpt,b0,index,i,j,k)

float beta[35][2], uncoded[35][2], s[35], midpt[35];
int b0,index,i,j,k;

{
  int loc_ij, loc_ik, loc_jk;      /* location of two-way
interactions */
  int loc_i, loc_j, loc_k, count; /* location of linear
terms */

  loc_ij = 0;
  loc_ik = 0;
  loc_jk = 0;
  loc_i = 0;
  loc_j = 0;
  loc_k = 0;

  /* UPDATE INTERCEPT TERM */
  uncoded[b0][1] = uncoded[b0][1] -
(beta[index][1]*midpt[i]*midpt[j]*midpt[k])/(s[i]*s[j]*s[k])
;

```



```

/* UPDATE THREE-WAY INTERACTION TERM */
uncoded[index][0] = (i * 100) + (j * 10) + k;
uncoded[index][1] = beta[index][1]/(s[i]*s[j]*s[k]);

/* LOCATE OR CREATE LINEAR TERMS */
for (count = 1; count <= nn; count++)
{
    if (uncoded[count][0] == i)
    {
        loc_i = count;
        uncoded[loc_i][1] = uncoded[loc_i][1] +
        (beta[index][1]*midpt[j]*midpt[k])/(s[i]*s[j]*s[k]);
    }
    if (uncoded[count][0] == j)
    {
        loc_j = count;
        uncoded[loc_j][1] = uncoded[loc_j][1] +
        (beta[index][1]*midpt[i]*midpt[k])/(s[i]*s[j]*s[k]);
    }
    if (uncoded[count][0] == k)
    {
        loc_k = count;
        uncoded[loc_k][1] = uncoded[loc_k][1] +
        (beta[index][1]*midpt[i]*midpt[j])/(s[i]*s[j]*s[k]);
    }
}
/* end for loop */

/* CHECK IF LINEAR TERMS DID NOT EXIST */
if (loc_i == 0)
{
    nn++;
    uncoded[nn][0] = i;
    uncoded[nn][1] =
    (beta[index][1]*midpt[j]*midpt[k])/(s[i]*s[j]*s[k]);
}
if (loc_j == 0)
{
    nn++;
    uncoded[nn][0] = j;
    uncoded[nn][1] =
    (beta[index][1]*midpt[i]*midpt[k])/(s[i]*s[j]*s[k]);
}
if (loc_k == 0)
{
    nn++;
    uncoded[nn][0] = k;
    uncoded[nn][1] =
    (beta[index][1]*midpt[i]*midpt[j])/(s[i]*s[j]*s[k]);
}

/* LOCATE OR CREATE TWO-WAY INTERACTION TERMS */
for (count = 1; count <= nn; count++)
{
    if (uncoded[count][0] == ((i * 10) + j))
    {
        loc_ij = count;
        uncoded[loc_ij][1] = uncoded[loc_ij][1] -
        (beta[index][1]*midpt[k])/(s[i]*s[j]*s[k]);
    }
}

```

```

    }
    if (uncoded[count][0] == ((i * 10) + k))
    {
        loc_ik = count;
        uncoded[loc_ik][1] = uncoded[loc_ik][1] -
(beta[index][1]*midpt[j])/(s[i]*s[j]*s[k]);
    }
    if (uncoded[count][0] == ((j * 10) + k))
    {
        loc_jk = count;
        uncoded[loc_jk][1] = uncoded[loc_jk][1] -
(beta[index][1]*midpt[i])/(s[i]*s[j]*s[k]);
    }
} /* end for loop */

/* CHECK IF INTERACTION TERMS DID NOT EXIST */
if (loc_ij == 0)
{
    nn++;
    uncoded[nn][0] = (i * 10) + j;
    uncoded[nn][1] =
-(beta[index][1]*midpt[k])/(s[i]*s[j]*s[k]);
}
if (loc_ik == 0)
{
    nn++;
    uncoded[nn][0] = (i * 10) + k;
    uncoded[nn][1] =
-(beta[index][1]*midpt[j])/(s[i]*s[j]*s[k]);
}
if (loc_jk == 0)
{
    nn++;
    uncoded[nn][0] = (j * 10) + k;
    uncoded[nn][1] =
-(beta[index][1]*midpt[i])/(s[i]*s[j]*s[k]);
}
} /* END FUNCTION THREEWAY */

```

Summary

The source code for PCRSIM was developed by Leeper and Meidt and modified as needed to complete this research effort (Leeper and Meidt, 1990:135-344). Original development resides in Regout2.c for the printing of the graphical displays and in Regout6.c in its entirety. Regout1a.c was created from the original Regout1 and modified to include variable decoding.

Appendix E: Optimization Background

Optimization

The optimization technique of RSM is the process of determining the optimum operating levels of the factors. Optimization of the factors is used in nearly all "process-orientated problems" (Myers, 1976:2). Myers gave the chemical industry as an example of the desire to determine optimum operating levels. It is often required to "find(ing) through experimentation conditions on the variables that give rise to desirable process yield" (Myers, 1976:2).

Optimizing the factors of the response function is an iterative process. As described by Kleijnen it involves selecting the appropriate experimental design; fitting the regression equation to determine the meta-model; finding the region of the stationary point using a "hill-climbing" algorithm such as steepest ascent; and classifying the stationary as a maximum, stationary ridge, saddle point, or rising ridge using canonical analysis (Kleijnen, 1974:80-81).

Steepest Ascent. The method of steepest ascent discussed by Box and Draper; and Myers is a method of sequentially moving in the direction of higher responses (Box and Draper, 1987:183; Myers, 1976:88). If the response

is to be minimize the method of steepest descent is applied. The difference between the methods of steepest ascent and steepest descent is the direction of movement. The direction of movement for steepest descent is opposite the direction for steepest ascent.

The method of steepest ascent is described by Box and Draper; Myers; and Mendenhall in the following manner. A fitted first-order model is a planar surface, which within the design region is representative of the actual surface. The slope of the planar surface is examined to determine the direction of steepest ascent, which is perpendicular to the slope. Movement is accomplished by selecting design points in the new design region and fitting the new first-order model after the additional experimental runs are made. This process is repeated until the design region is in the vicinity of the optimum. The optimum region can be detected by increasingly smaller movement steps (Box and Draper, 1987:182-187; Myers, 1976:88-90; Mendenhall, 1968:272).

Canonical Analysis. When in the vicinity of a "near-stationary region" (Box and Wilson, 1951:23) Baasel states that a second-order factorial design should be used to classify the stationary region (Baasel, 1965:152). Box and Wilson recommend canonical analysis of the second-order meta-model to classify the region.

Canonical analysis is a method of rotating the axes of

the factors along the major and minor axes of the contour system (Box and Draper, 1987:332-334; Myers, 1976:72-74). This rotation transforms the second-order meta-model into a quadratic function without interaction terms called the A canonical form. Classification of the stationary region and interpretations of the response surface can be made from the A canonical form.

The coefficients of the quadratic terms in A canonical form are the eigenvalues of the symmetric quadratic matrix of second-order β parameters. The sign of the eigenvalues is used to classify the stationary point: all negative eigenvalues indicate a maximum, all positive eigenvalues indicate a minimum, and differing signs indicate a saddle point.

Myers discussed the interpretation of the response surface that can be gained by examination of the eigenvalues. The magnitude of an eigenvalue is an indication of the sensitivity of the estimated response to changes in that direction (Myers, 1976:75). Box stated that "if one of the coefficients [eigenvalues] is small in magnitude compared with the others then the surface is attenuated along the axis corresponding to the small coefficient" (Box, 1954:36). Movement along the attenuated axis yields smaller changes to the estimated response than movement along the other axes.

Cornell also discussed the importance of eigenvalues for interpreting the response surface (Cornell, 1984:21-23). A ridge system can be detected if one eigenvalue is close to zero and all others are negative. The location of the stationary point identifies the type of ridge system. If the stationary point is within the design region then the ridge system is stationary; if the stationary point is not near the design region then the ridge system is rising (Myers, 1976:75-78, Cornell, 1984:21-23). A stationary ridge system is important to determine because

it is thus possible to indicate alternative sets of conditions at which almost equal responses would be expected and so to find the most satisfactory compromise for auxiliary responses in the chosen process. (Box and Wilson, 1951:24)

Confidence Region for Optimum Levels. The determination of the optimum for the response surface "can only be an approximate one, since the data on which it is based are approximate" (Box and Draper, 1987:126). Box and Hunter developed a "confidence region for the stationary point on a fitted second-degree surface" (Box and Hunter, 1954:195) to determine limits on the error of the stationary point location.

To calculate the confidence region for the stationary point the following assumptions must hold:

1. The errors are distributed multinormally.
2. The variance-covariance matrix of the least-squares estimates is known.
3. An estimate s^2 of σ^2 is known. (Box and Hunter,

1954:190, 195)

Equation (20) provides the confidence region.

$$\left| \begin{array}{cc} s^2 p F_{\alpha} & \mathbf{Y}' \\ \mathbf{Y} & \mathbf{V} \end{array} \right| \quad (20)$$

where

s^2 = estimate of σ^2
 p = number of parameters estimated
 \mathbf{Y} = response vector
 \mathbf{V} = variance-covariance matrix

If the canonical analysis "indicate that the fitted surface possesses a maximum, and the confidence region is closed, we may assert that this is a confidence region for a true maximum" (Box and Hunter, 1954:196).

Bibliography

- Baasel, William D. "Exploring Response Surfaces to Establish Optimum Conditions," Chemical Engineering: 72 147-152 (October 1965).
- Box, G. E. P. "The Exploitation and Exploration of Response Surfaces: Some General Considerations and Examples," Biometrics: 10 16-60 (March 1954).
- Box, G. E. P. and D. W. Behnken. "Some New Three Level Design for the Study of Quantitative Variables," Technometrics: 2 455-475 (November 1960).
- Box, George E. P. and Norman R. Draper. Empirical Model-Building and Response Surfaces. New York: John Wiley and Sons, Inc., 1987.
- Box, G. E. P. and J. S. Hunter. "A Confidence Region for the Solution of a Set of Simultaneous Equations with an Application to Experimental Design," Biometrika: 41 190-199 (1954).
- Box, G. E. P. and K. B. Wilson. "On the Experimental Attainment of Optimum Conditions," Journal of Royal Statistical Society Series B: 13 1-45 (November 1951).
- CLIPPER Manual. Los Angeles: Nantucket Corp., 1988.
- Cornell, John A. Volume 8: How to Apply Response Surface Methodology. Milwaukee: American Society for Quality Control, 1984.
- Hillier, Frederick S. and Gerald J. Lieberman. Introduction to Operations Research (Fourth Edition). Oakland, Ca: Holden-Day, Inc., 1986.
- King, David. Current Practices in Software Development A Guide to Successful Systems. New York: Yourdon, Inc., 1984.
- Kleijnen, Jack P. C. Statistical Techniques in Simulation Part I. New York: Marcel Dekker, Inc., 1974.
- Leeper, Capt David M. and Capt Gregory J. Meidt. A Micro-Computer Based Decision Support System for Response Surface Methodology. MS Thesis, AFIT/ENS/GOR90M-12. School of Engineering, Air Force Institute of

- Technology (AU), Wright-Patterson AFB OH, March 1990.
- Meidt, Gregory J. and Kenneth W. Bauer. "PCRS: A Decision Support System for Simulation Metamodel Construction," AFIT ENS Working Paper 90-6, 1990.
- Mendenhall, William. Introduction to Linear Models and The Design and Analysis of Experiments. Belmont, Ca: Wadsworth Publishing Company, Inc., 1968.
- Myers, Raymond H. Response Surface Methodology. Virginia Polytechnic Institute, 1976.
- Pressman, Roger S. Software Engineering A Practitioner's Approach (Second Edition). New York: McGraw-Hill Book Company, 1987.
- Stevens, Roger T. Graphics Programming in C A Comprehensive Resource for Every C Programmer. Redwood City, Ca: M&T Publishing, 1988.